NPS-ME-97-008

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTIPLE AUTONOMOUS VEHICLES FOR MINEFIELD RECONNAISSANCE AND MAPPING

by

Jack A. Starr

December, 1997

Thesis Advisor: Anthony J. Healey

Prepared for:
Naval Surface Warfare Center, CSS
Panama City, Florida 32407

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943

Rear Admiral Marsha Johnson Evans
Superintendent

This thesis was prepared in conjunction with research sponsored in part by Naval Surface Warfare Center, Coastal Systems Station, Panama City, Florida, under N0001497WX30039.

Released by:

[illegible]

David Netzer, Associate Provost
And Dean of Research

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. **REPORT DATE** **December 1997**	3. **REPORT TYPE AND DATES COVERED** **Master's Thesis**

4. **MULTIPLE AUTONOMOUS VEHICLES FOR MINEFIELD RECONNAISSANCE AND MAPPING**	5. FUNDING NUMBERS N0001497WX30039
6. AUTHOR(S) Jack A. Starr	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER NPS-ME-97-008
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Surface Warfare Center, Coastal Systems Station, Panama City, FL 32407-7001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

ABSTRACT *(maximum 200 words)* The development of numerical search modeling for Autonomous Search Vehicles (ASV's) is an essential tool for development of ASV strategy using groups of small, crawling vehicles. Reconnaissance of surf-zone bottoms for mines and obstacles, as well as providing an environmental mapping capability, is the objective. These models allow numerical simulations to be conducted that determine the relationships between search times, target and obstacle sensing radius, vehicle speed and numbers of vehicles using simple, preprogrammed search strategies. The results from these simulations on intial models can then be used to determine the overall system performance.More complex models can then be developed using search strategies that include directed search, avoidance behaviors, networking and mapping with sufficient navigational accuracy. With sufficient information on ther behavior of these vehicles, the ultimate goal of providing an autonomous reconnaissance and neutralization capability in very shallow water and surf zones can be realized.

14. SUBJECT TERMS: ASV, Surf Zone Reconnaissance Mission, Simulation, State-Based Robotics	15. NUMBER OF PAGES 131
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

MULTIPLE AUTONOMOUS VEHICLES FOR MINEFIELD RECONNAISSANCE AND MAPPING

Jack A. Starr
Lieutenant, United States Navy
B.S., Oregon State University, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December, 1997

Author: Jack A. Starr
Jack A. Starr

Approved by: [illegible]
Anthony J. Healey, Thesis Advisor

[illegible]
Terry R. McNelley, Chairman
Department of Mechanical Engineering

ABSTRACT

The development of numerical search modeling for Autonomous Search Vehicles (ASV's) is an essential tool for development of ASV strategy using groups of small, crawling vehicles. Reconnaissance of surf-zone bottoms for mines and obstacles, as well as providing an environmental mapping capability, is the objective. These models allow numerical simulations to be conducted that determine the relationships between search times, target and obstacle sensing radius, vehicle speed and numbers of vehicles using simple, preprogrammed search strategies. The results from these simulations on initial models can then be used to determine the overall system performance. More complex models can then be developed using search strategies that include directed search, avoidance behaviors, networking and mapping with sufficient navigational accuracy. With sufficient information on the behavior of these vehicles, the ultimate goal of providing an autonomous reconnaissance and neutralization capability in very shallow water and surf zones can be realized.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to acknowledge the financial support of the Naval Surface Warfare Center's Coastal Systems Station, Panama City, for funding this thesis. I would also like to thank Chuck Bernstein for his support and information regarding the details of the Surf Zone Reconnaissance Mission, and for sharing his enthusiasm on the potential of these robotic vehicles.

I want to express my everlasting gratitude to Professor Healey for all his assistance and patience on this thesis. Professor Healey is a great example of the advice to pick the thesis advisor and not the thesis topic. He has provided me with the resources to solve some very interesting problems, and I certainly appreciate the guidance regarding many of the technical issues that face the potential users of autonomous vehicles. A big thanks to Mr. Joung Kim for all his expertise in code development and making the numerical simulations a success. It was a truly rewarding and professional experience.

Lastly, and most importantly, I would like to thank three people who have always inspired me in my educational endeavors: Jim Mitchell, who got this all started; my Mother, who always believed I'd end up here; and most importantly, my wife, Sarah, for all her love and support as I wind my way down this interesting road of life.

I. INTRODUCTION

A. BACKGROUND

The use of multiple, small robotic vehicles for performing reconnaissance, marking and clearing of mines and minefields, as well as clearing unexploded ordinance from areas of interest, is currently an active pursuit of the Navy's Explosive Ordinance Disposal (EOD) Research and Development Department. Known as a Basic Unexploded Ordinance Gathering System (BUGS), these small robots are being designed to assist EOD technicians whose responsibilities include entering battlefields and test-firing ranges to clear "improved munitions", as well as similar systems for performing the reconnaissance mission that supports the clearing of approach lanes to the beach for amphibious force landings. For the land-based operations, the goal is to improve safety and performance with these smart machines, with the EOD technician having the capability to instruct the vehicles to transit an open area, while performing the necessary obstacle avoidance, and pick up pieces of ordinance or place charges that can be detonated upon command. This range remediation and battlefield-clearance aspect of these small, autonomous robots will also reduce the EOD squad's time-consuming task of clearing the affected areas after hostilities cease. In the Surf Zone Reconnaissance Program, the goal is to provide a shallow-water mine, obstacle and environmental mapping capability to the amphibious warfare commanders to facilitate amphibious attack and clearance operations. In addition to the reconnaissance mission, the technologies to use groups of small robotic vehicles to mark ordinance in the surf zone and conduct mine neutralization missions are also being developed. By eliminating the human from the immediate area, the EOD technicians will be able to formulate decisions from remote sites and determine which items to remove. There are many other potential advantages to using groups of small, crawling BUGS to search the battlefields and surf zone bottoms. Close-range sensing may offer better classification potential than long-range sensing, particularly against well-buried mines. Multi-vehicle systems can provide search redundancy and improved system reliability. Once deployed in the surf zone environment, a group of Autonomous Search Vehicles (ASV) could be covert, and it could operate during the day or night and in adverse weather. With the current technology involving essentially a brute-force approach, the promise of autonomous BUGS would provide a method with more stealth and the ability to prevent loss of life and limb.

B. SCOPE OF THESIS

This thesis is intended to document and detail the developments of numerical simulation studies to examine strategies for the combined usage of many low-cost BUGS in land-based and surf-zone environments. These simulations are, in particular, developing performance data for the use of BUGS in pick-up and carry-away (PUCA) clearance operations for the land-based simulations, and performance data on the reconnaissance capabilities for the surf-zone mission. The work presented for the land-based simulations of the autonomous vehicles builds on work done by Healey and Kim in [1] for the Naval Explosive Ordinance Disposal Technology Division, Indian Head, MD., conducted at the Naval Postgraduate School. For the land-based scenario, the ultimate purpose of the work is to develop the low-level (per BUGS) and high-level (fleet) control strategies needed to implement a fleet of BUGS vehicles to perform the remediation of large areas of land in reasonable amounts of time. For the surf-zone reconnaissance mission, the goal is for the small fleet of vehicles to be placed in the amphibious approach lane from the seaward side, and perform the necessary locating and marking of the unexploded ordinance (UXO) that is to be disposed. In short, these numerical simulations will allow the user to determine the most optimal use of his autonomous robotic assets and to determine the clearance and reconnaissance times to be expected for the various modeled scenarios.

Chapter II will cover the derivation and development of the mathematical modeling of search algorithms and sensor packages and how they relate to the overall simulation. Chapter III discusses obstacle avoidance modeling and how it is implemented into the simulation code. Chapter IV develops the vehicle dynamics model and its interaction with the sensor and obstacle avoidance routines. Chapter V will discuss the results of numerical simulations performed to compare various scenarios and vehicle combinations for the land-based and surf-zone reconnaissance models. Chapter VI will summarize this thesis, present the conclusions, and make recommendations for further simulation code development.

II. SEARCH AND SENSOR MODELING

A. GENERAL

The intent of this chapter is to present the derivation and modeling of the search and sensor package developed in the numerical simulations of the autonomous vehicles conducting both PUCA operations in the UXO field and threat identification in the surf zone. The coverage problem is, without a doubt, one of the primary challenges faced by an autonomous vehicle attempting to conduct a thorough search of the UXO field or identifying mine-like objects in the approach lane for a potential amphibious landing. Specifically, the type and method of search has spurred on much research and accompanying theory, with good examples being the work conducted by A.P. Washburn [2] and the research by Healey and Kim in [3]. The bulk of the research has concentrated on exhaustive searches and the use of random search. The exhaustive search, much like "mowing the lawn", is a method sound in theory but impractical in actual use due to expected navigation errors on the vehicle. When one considers the complicated dynamics of the surf zone in a reconnaissance mission, and the lack of access to accurate navigation data, the shortfalls of the exhaustive search method become even more evident As the performance degrades on an exhaustive search, research shown by [3] indicates that even a seven-degree error in heading data produces loss of area coverage to the point where a random search could have been cheaper and equal in performance. The random search, however, due to its preprogrammed random search behavior within the UXO field and approach lane, doesn't require the level of navigational accuracy, and therefore has proven to be a likely candidate for the search algorithm of fleets of bugs conducting the minefield searches and reconnaissance missions. The simulations examined in this thesis were conducted using the random search method when the vehicle enters the search zone, but assumes accurate navigation is possible. Additionally, the sensor modeling becomes a key component in the modeling process, especially when contemplating the expected sensor detection radius and probability of detection. Additional areas of concern for the sensor selection concern the issue of sensor range outperforming the ability of the vehicle to process the data and conduct the required target identification and obstacle avoidance. Therefore, the main thrust of the simulations, for sensor purposes, was to determine the minimum, and possibly maximum, capabilities of the sensor systems required to accomplish target disposal and identification in reasonable times. By varying the sensor radius and probability of detection factors with various target and obstacle scenarios, this goal was accomplished and the results can be compared to find an effective sensor for the given mission.

B. SEARCH MODEL - UXO PUCA

As stated by Washburn, the rate of detection of UXO using a directed search is given by the following [2]:

$$\dot{q}(t) = U(2r)pN\left(\frac{n_0}{A}\right) \qquad (2.1)$$

where **$\dot{q}$(t)** is the expected rate of new detections, **U** is the speed of the vehicle, **r** is the sensor's radius of detection, **p** is the sensor's probability of detection when a UXO is within the detection radius, **N** is the number of bugs, **A** is the area to be searched, and **no** is the initial number of

UXO. For purposes of our simulations and models, statistical independence and uniform probability distribution of UXO are assumed. Solving the equation above for **q(t)**, we find that the solution to the exhaustive search is a linear function of time, where **U(2r)pN(no/A)** is the slope of the linear solution varying with time, **t**. So, for an exhaustive search, the clearance would be complete at **t=A/U2rpN**. Since our simulations are modeling a random search strategy, the rate of new detections is a function of the expected number of UXO which is reduced by the number of targets already detected, so that

$$\dot{\bar{q}}(t) \quad = \quad U(2r)pN\left(\frac{n_0 - \bar{q}(t)}{A}\right) \qquad (2.2)$$

This leads to the following equation for the expected number of UXO detected as a function of time

$$\bar{q}(t) \quad = \quad n_0\left(1 - e^{\alpha t}\right) \qquad (2.3)$$

where α **=[U(2r)pN/A]** is called the characteristic clearance rate.

In our land-based simulation, the area searched was assumed to be 30m square and the BUGS vehicle knows the orientation of the square with respect to compass headings and also has the ability to determine which side is being approached when it is close, that is, less than a meter. This can be accomplished using four electronic fences, each emitting a different

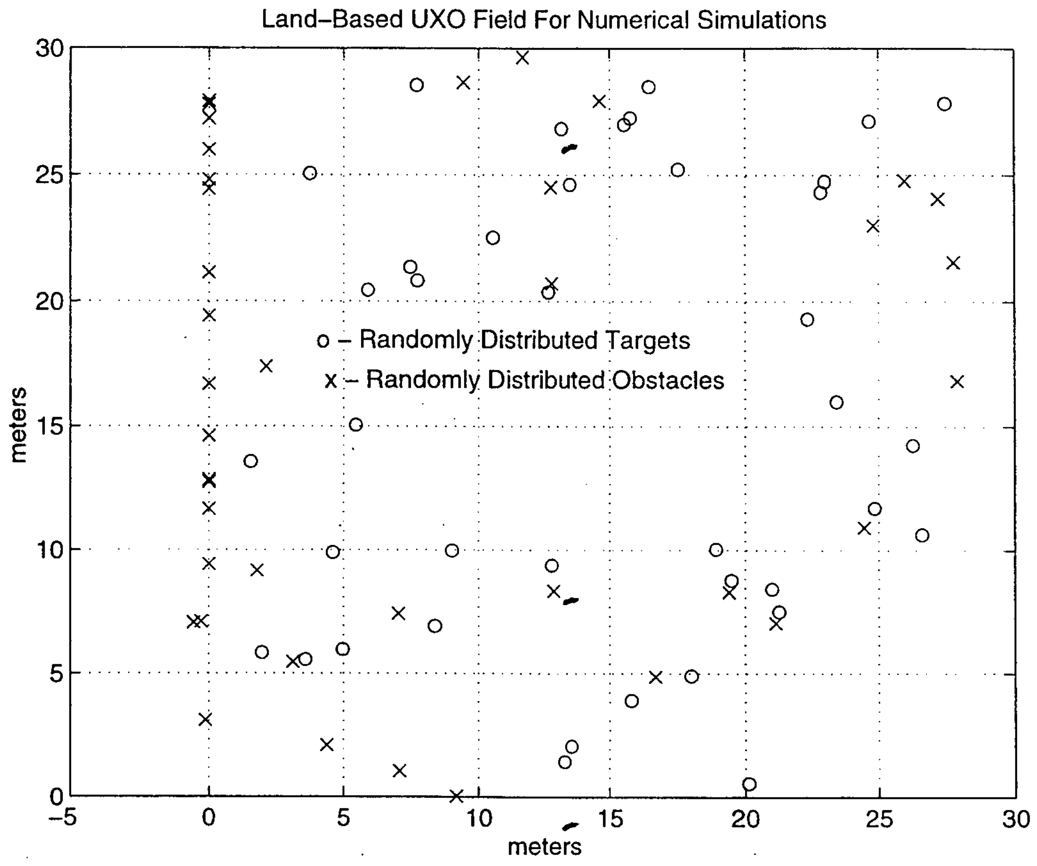


Figure 2.1 Simulated Land-Based UXO Field For Numerical Simulations

frequency. Figure 2.1 shows a typical field of UXO, for numerical simulation purposes, with randomly distributed targets for PUCA operations. After the fleet of BUGS is placed within the fence, a heading bias is established, and is set perpendicular to the nearest side and points into the search area.

1. Randomization of Vehicle Heading

The controller randomizes the search by selecting a steering law expressed as

$$\psi_{command} = \psi_{bias} + \psi_{random} \qquad (2.4)$$

where ψ random for the land-based simulation was uniformly distributed over **$[-\pi/2, \pi/2]$**. Periodically, or approximately every two seconds in the simulations, a new heading will be chosen in an identical manner. Most importantly, any time the BUGS vehicle approaches within a meter of any of the fences, its bias heading is changed to reflect the BUGS back into the search area. The idea is to allow a very crude compass and a simple radio frequency receiver to meet the navigation requirements of the BUGS autonomous mission and still keep the costs low enough to support the numbers of vehicles required to conduct the clearance operations. Following searches that result in the vehicle detecting and picking up a target, the vehicle changes it's heading control basis to the drop-off area. The vehicle will then navigate to the drop-off area, avoiding both obstacles and targets which are not yet acquired while enroute to the disposal place. When the vehicle enters the drop-off area, it drops off the target and reenters the field to continue searching. As mentioned before, a new heading is chosen periodically. Any time the BUGS

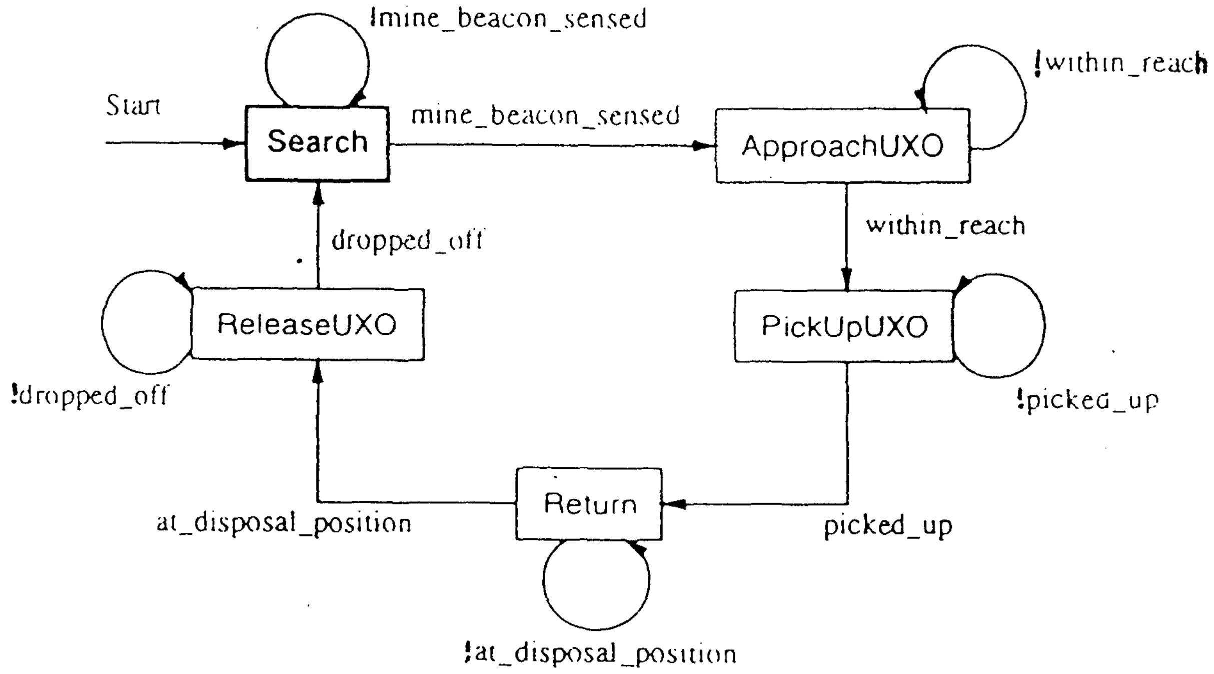


Figure 2.2 State Diagram For Search Algorithm, EOD Land-Based BUG

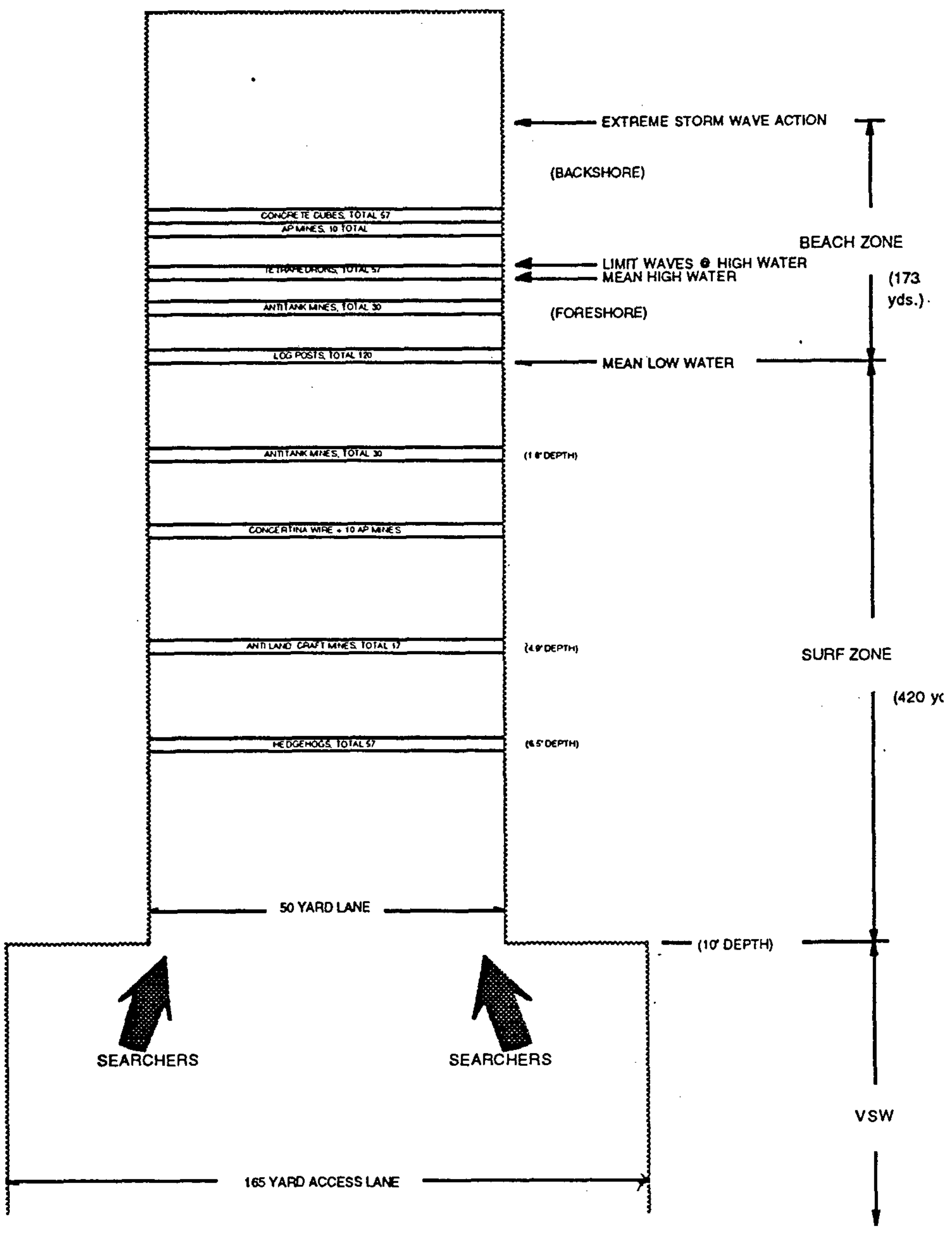


Figure 2.3 Approach Lane For Surf Zone Reconnaissance Mission

vehicle meet any of the fences, its bias heading is changed to reflect the BUGS back into the search area. A state diagram showing the search algorithm for the EOD land-based BUG is shown in Figure 2.2.

C. SEARCH MODEL - SURF ZONE RECONNAISSANCE

In the Surf Zone Reconnaissance Mission, the approach lane is a 420 by 50 yard area with mines and obstacles laid uniformly in "bands" as shown in Figure 2.3. The vehicles are released from two points, specifically, at the seaward corners of the approach lane to be searched. This arises from assuming a two-buoy navigation system, with the buoys packaged into the deployment pods for the vehicles. A simple travel-to-zone strategy is utilized, where the lane is divided into a number of equal area rectangular zones parallel to shore, and the same number of searchers is dispatched to each zone. The searchers are released two at a time, one from each drop-off location every three seconds, with the searchers headed for the beachward zones released first and the adjacent zones filled as the beachward zones fill up with vehicles. The searchers will be programmed to travel to their designated search zones using a uniform random heading in the zone's general direction, so as to cause them to arrive in a dispersed pattern in the zone. Additionally, the searchers will not begin hunting for mines until they arrive within their designated search zones.

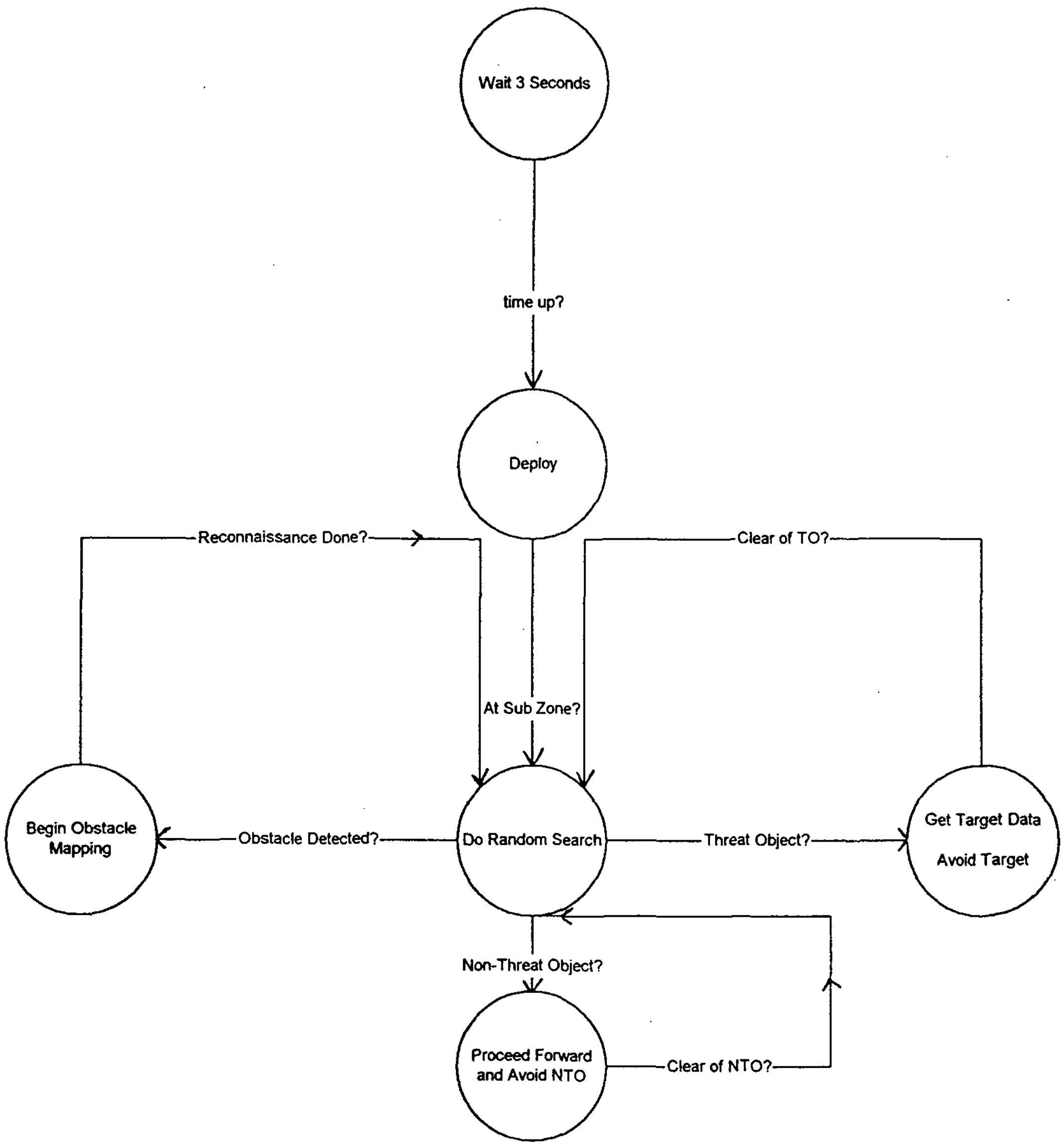


Figure 2.4 Surf-Zone Reconnaissance Vehicle Search-Mode State Diagram

Examination of the Surf-Zone search algorithm shows that the autonomous vehicles have a search pattern that is designed to maximize their effectiveness as a reconnaissance tool. Figure 2.4 shows the state diagram representing the simulated search algorithm for the Surf Zone Reconnaissance vehicle. Initial entry in the zone requires the vehicle to drive on its initial random heading for 50 seconds or until a target or obstacle is encountered. After 50 seconds or initial contact, whichever comes first, the searchers all begin a pseudorandom search strategy within that zone that is a repeating sequence of forward motion at search speed, **Vf**, for a time interval, **Tf**, that is between heading changes that can be selected by the operator. As will be discussed in Chapter IV at length, simulations for the Surf Zone Reconnaissance and EOD PUCA missions indicate the faster identification rates are achieved with a longer time between heading changes, **Tf**. Following the interval, **Tf**, a position check is carried out, typically one second, followed then by a random turn of +-90 or +-120 degrees. The searchers in the surf zone also have an onboard compass and are preprogrammed with the beach heading. The search continues on in this way until the mission is complete, which usually implies the battery life of the vehicle is exhausted. At this point, it is presumed that the vehicles in each of the various zones have had an opportunity to make multiple visits to the targets and obstacles within the zone boundaries, with the locations and threat evaluations by the vehicles passed on to the amphibious commanders. The searchers will encounter every object, that is, mines, as well as artificial and natural obstacles which pass within their sensing radius. All objects, for purposes of the simulations, will be assumed detected and undergo classification. During an encounter, the searcher will have asingle opportunity to classify the object as either a Threat Object (TO), or as a Non-Threat Object (NTO). The classification process, for purposes of the simulation, takes 3 seconds. During this time, the searcher will not move. Following classification, if the object is classified a TO, then the searcher will sit for **Td** seconds, which will represent the time required to obtain a position fix and report back the contact information.

D. SENSOR MODEL

The sensor model, as far as the simulation studies have been developed, have assumed a variety of sensor radii that have allowed various scenarios to be studied. In the land-based models examined, the radius of detection of the primary detection sensor was .3810 m. The assumed probability of detect has also been varied in the land-based and surf-zone scenarios, with assumed probability of detects ranging from 1 to as low as .6 for the sensor and scenario under study. The probability of detect has been assumed to be uniformly distributed in a circle of the defined radius from the center of the vehicle. This is the model that all the land-based and surf-zone reconnaissance models have been built on for the numerical studies carried out in this thesis.

In the numerical simulations conducted on the land-based PUCA operations for the Navy EOD vehicle with random search methods, the area to be searched is assumed to be square and the BUGS vehicles know the orientation of the rectangle with respect to compass headings and also have the ability to determine which side is being approached. This can be accomplished with four electronic fences emitting at different frequencies. For purposes of the simulation, if any target lies within the detection radius of any vehicle, then that target is assumed to be acquired by that vehicle. Following that search, the vehicle is assumed to change it's heading control basis to a homing basis, the drop-off area, where the PUCA operation is completed and a new random search starts.

In the surf-zone simulation, classification using the modeled sensor will be based on generating a uniform random variable scaled to between 0 and 100 percent. This number will be compared against the Probability of Correct Classification (PCC) parameter which will be set for the current search run. If the random number is less than the PCC, the object, whether TO or NTO, will be considered to be correctly classified, and vice versa. The simulation notes each valid classification of a TO and each false classification of a TO, or false alarm, and keeps a count of all encounters, as well.

III. OBSTACLE AVOIDANCE

A. GENERAL

There are many autonomous-vehicle, obstacle-avoidance schemes possible, with the one used depending, to a large degree, on the type and availability of sensors that provide information about the obstacles in question. In the simulation of autonomous vehicles, two different approaches to obstacles avoidance are used. First, a state-based, obstacle-avoidance method is simply to stop, backup, turn, go forward, turn back, and continue. This approach is used by the Foster Miller [3] and is shown in Figure 3.1. Alternatively, a behavior-based avoidance method weights the steering commands determined by obstacle avoidance behavior with commands generated for "homing" or "transit to target" through a prioritizer which can change or arbitrate between behaviors according to the relative importance of each behavior. This approach is shown Figure 3.2. Some sensors for obstacle avoidance that might be used include current-induced torque sensors in the wheel motors, mechanical bumpers, tactile whiskers, IR detectors and sonar belts. The disadvantages of bumpers or torque sensors is that very little information about the obstacle in front of the vehicle is available except for its presence. As mentioned previously, a basic state-based avoidance scheme is to back up, turn (100 degrees, for example) , go forward a prescribed distance (one meter, for instance), turn into target direction again and test for contact. The idea is that eventually this scheme will divert the vehicle sufficiently far away from the obstacle to allow progress. This simple scheme works well in some cases, but would not in others, such as a blind alley scenario, or the case of the extremely long obstacle. The state-based scheme

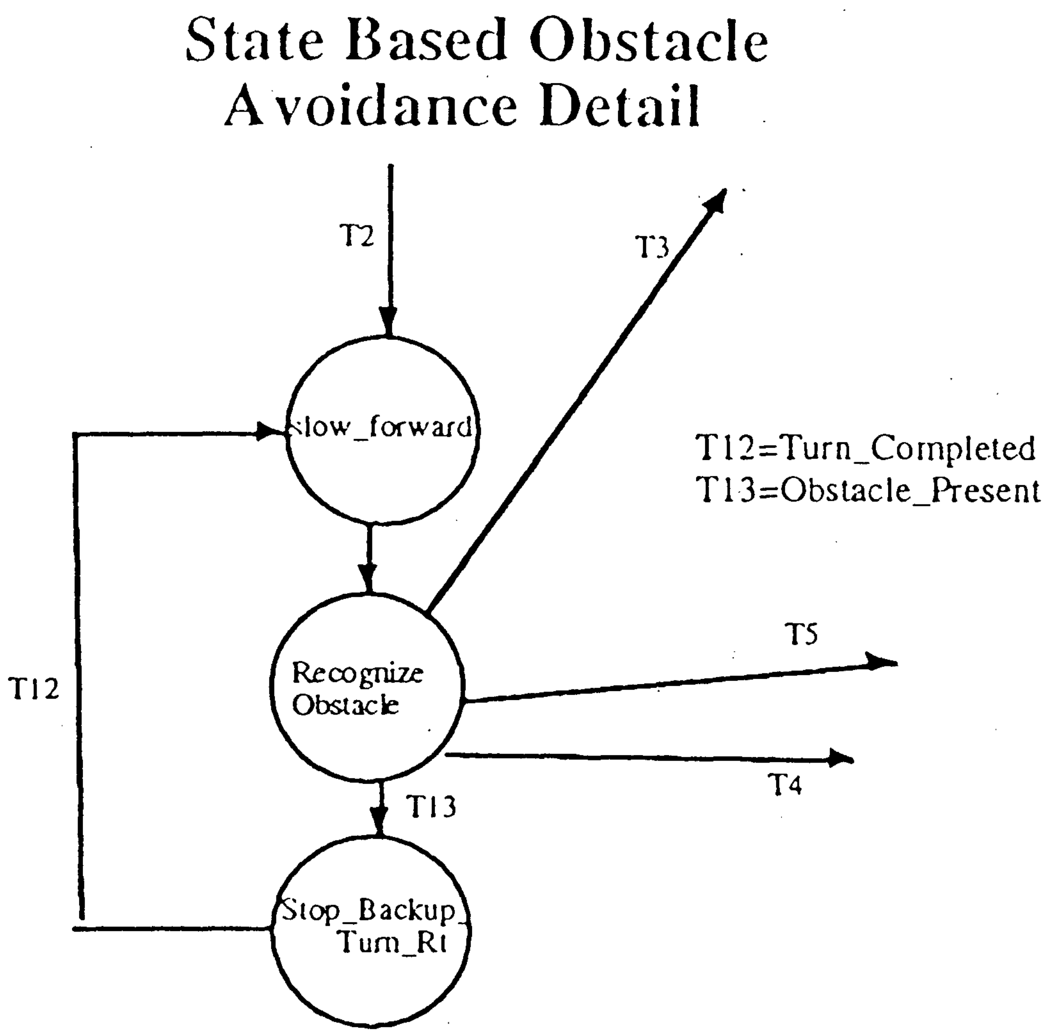


Figure 3.1 State-Based Obstacle Avoidance Detail

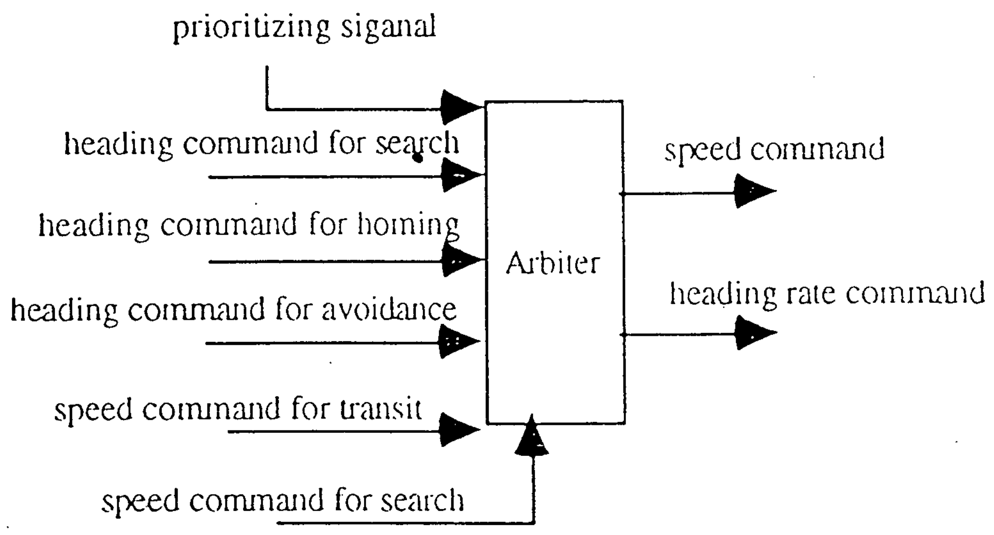

prioritizing siganal
heading command for search
heading command for homing
heading command for avoidance
speed command for transit
speed command for search
Arbiter
speed command
heading rate command

Figure 3.2 Behavior-Based Obstacle Avoidance Detail

is slower and prone to trapping more frequently, with the principal disadvantage in this scenario being that the obstacle avoidance scheme is slow to return the vehicle to its primary path of progress. Better results can be provided if the element of directionality can be provided with a more complex scheme. The behavior-based control is smoother in operation and will yield faster and more reliable obstacle avoidance. Figure 3.3 shows a generic scheme that has a 90-degree sector in front of the vehicle that is divided into a right and left subsection that can distinguish which side of the vehicle the obstacle exists. The vehicle's back and forth problem is solved by restricting the obstacle-sweeping sensor's angle. The obstacle sensor's sweeping angle is now reduced to +- 45 degrees from the vehicle's primary heading direction. A much more reliable and resistant method, this tends to avoid the trapping around complex obstacles, with more sectors allowing the vehicle to become a reliable analog to a behavior-based vehicle. The sectoring of the detection of obstacles essentially allows for smoother obstacle avoidance. Figures 3.4 and 3.5 compare the state-based and behavior-based obstacle avoidance schemes with simulation traces of the vehicles, with the behavior-based model showing significantly more effective and efficient obstacle avoidance as it moves around in the UXO field.

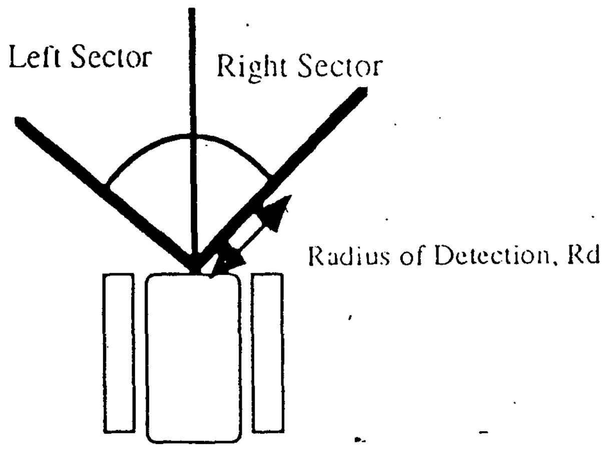


Figure 3.3 Generic Scheme- 90 Degree Sector

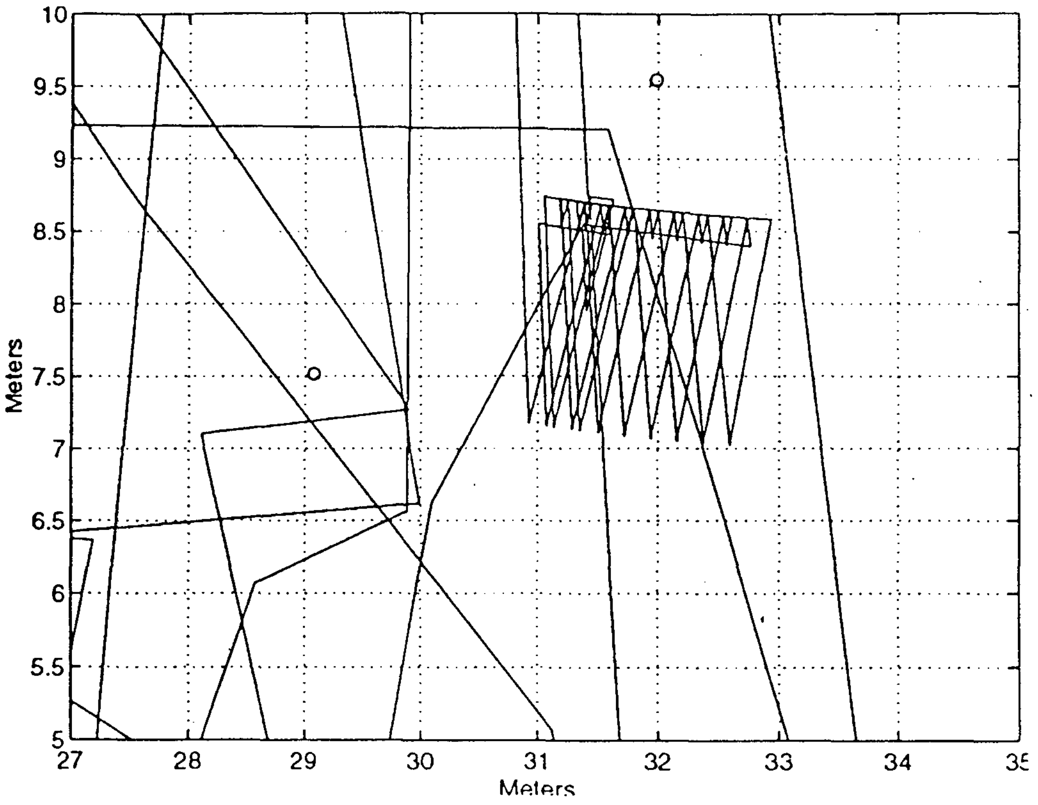


Figure 3.4 State-Based Obstacle Avoidance Simulation Trace

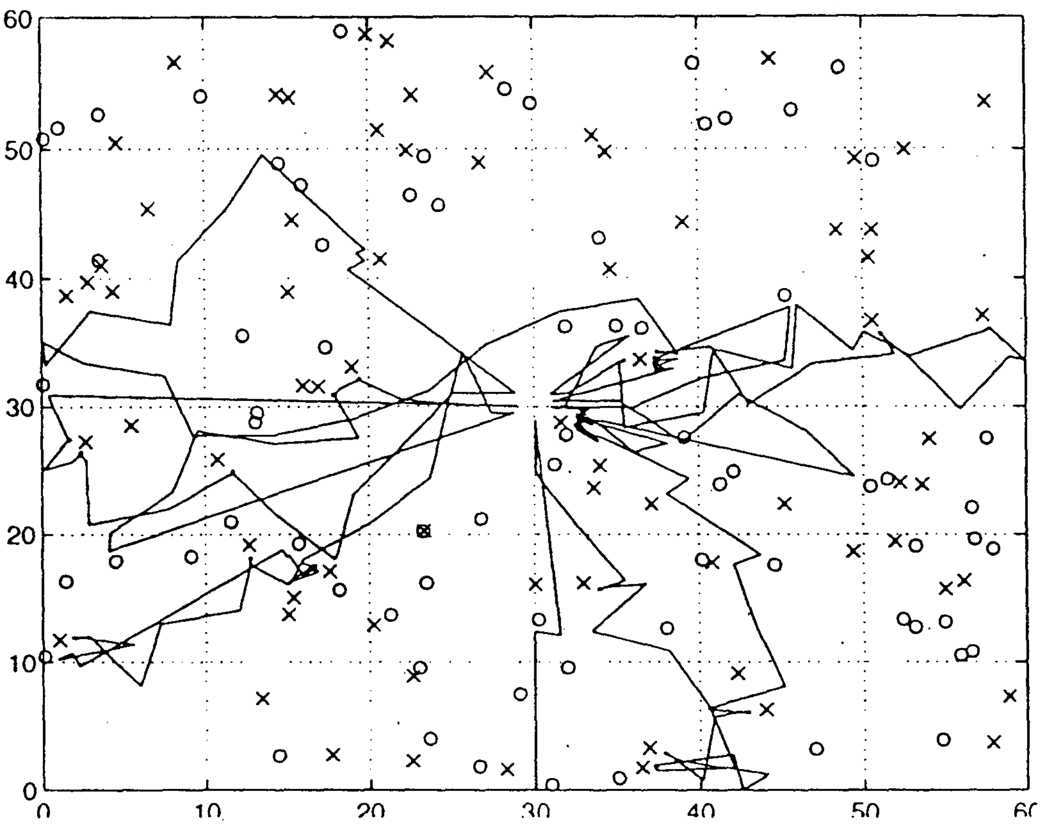


Figure 3.5 Behavior-Based Obstacle Avoidance Simulation Trace

B. THE EOD LAND-BASED MODEL

The EOD model utilizes the following obstacle-avoidance pseudo algorithm in the behavior-based simulation model:

```
While obstacle detect radius < Rd
        Rotate left if detect lies in right sector
        Rotate right if detect lies in left sector
end
        Move forward one increment step
If BUG turned left and moved by one full step, turn right
If BUG moved 1 step but did not turn left, head to goal point
Continue Searching or Dropping if BUG is within Goal Neighborhood
```

This behavior does not include the vehicle backing up, is active at all times, and provides a boundary-following characteristic in a clockwise fashion around an obstacle while overcoming the trapping problem of the simpler, state-based scheme. In the PUCA scenario, one could describe the obstacle-avoidance behavior of the vehicle in terms of a finite state machine, Figure 3.6. Where there is no obstacle to avoid, the system stays in its operational mode. When the BUGS vehicle is not performing PUCA operations and is not avoiding obstacles, the system remains in the search mode that was described in the previous chapter. In this EOD land-based scenario, obstacle

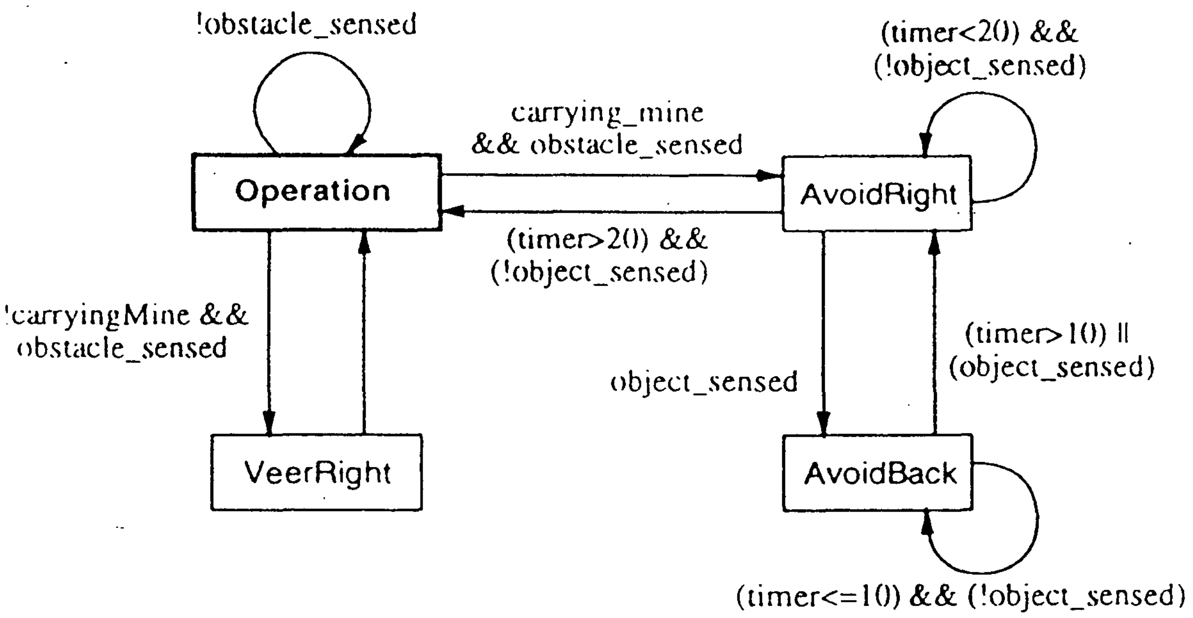


Figure 3.6 PUCA Scenario Finite State Machine

avoidance is handled in two ways, depending on which mode, Search or PUCA, the vehicle is in. When the BUGS vehicle is in the search mode and an obstacle is detected, a new heading is chosen (90 degrees to the right) as indicated by the Veer Right State. This is done for a couple of reasons. It is much easier than trying to navigate around the object and then attempting to reacquire the previous heading, and it also reinforces the randomness of the path. If the turn were to move the BUGS too far, say, 90 degrees or more from its bias heading, the next periodic heading, or the fence, will correct this. Additionally, a BUGS vehicle in the UXO field must be able to avoid obstacles during its return to the disposal point. Since this is not a random walk at this point, a different avoidance behavior is implemented on the simulation runs. In this case, the vehicle turns to the right (Avoid Right) and tries to go that direction a certain amount of time. If it does not encounter an obstacle, it will then return to its original heading towards the disposal point. However, if it manages to encounter another obstacle during the "Avoid Right" mode, it will turn another 90 degrees to the right and move backwards from the original heading (Avoid Back). From this point, the vehicle will try to move right and then return to its original heading towards the disposal point.

C. THE SURF-ZONE RECONNAISSANCE MODEL

The Surf Zone Reconnaissance Mission looks at the topic of obstacle avoidance in a very different way. As the mission is to detect mine-like targets and obstacles that may prove to be hazardous to the amphibious landing, the challenge is to not to avoid the obstacles that may interfere with mine identification, but to identify the obstacles, as well, by navigating around them and determining their physical size. Since the presence of well-placed obstacles on the beaches may slow down the landing forces, it is imperative to know the location and size of these man-made obstacles, and the autonomous vehicles obstacle avoidance routines not only keep the BUGS from running into the obstacles, they also help them to determine their location and size. In short, the searchers have an obstacle-avoidance mechanism enabling them to circumscribe obstacles with diameters greater than the vehicles search-circle diameter. Based on the location and size, the obstacle can be evaluated by the warfare commanders as a threat, or even possibly a mine-like object that wasn't detected as such by the primary target sensor. If a target-like object has been classified as a Non-Threat Object (NTO), and the report-back has been finished, the searcher will move forward until the object is out of the circle defined by the search radius (Rs). When the object is out of the circle of detection, it will continue to move forward for another distance Rs. If for some reason the searcher collides with the newly-classified target, it will sense this, and make a random turn and attempt to travel away from the object, repeating this procedure until it is successful. It is possible for the same searcher, by random searching, to encounter the same object multiple times. In the case of the obstacle avoidance and identification routine, the vehicle will "remember" the objects it has navigated around and reported the positions of, and therefore, will only back and turn from the reported "obstacles" not identified as a target by the primary mine sensor.

The routine that the Surf-Zone Reconnaissance Vehicle simulation uses to conduct the obstacle identification mission is an algorithm that essentially combines the primary obstacle sensor in coordination with a simple state-based behavior response that allows the vehicle to navigate around the obstacle and identify its approximate physical diameter and location in the approach lane. The pseudo algorithm that describes the obstacle-identification behavior that results after a circumnavigation of the obstacle can be described as:

```
While Sensor Detects Obstacle
        Stop
        Sensor determines shortest distance to obstacle
        Turn left 50 degrees (from the heading to the closest point of obstacle)
end
        Drive Forward for one increment step
        Stop
        Turn Right 90 degrees
```

For purposes of visualization of the discussion, assume a circular obstacle. The vehicle approaches the obstacle conducting reconnaissance at its search speed until the primary obstacle-detection sensor detects its presence. At this point, the vehicle stops and the vehicle ascertains the shortest distance to the newly-found obstacle based on its applicable sensors. After this distance has been ascertained, the vehicle turns left 50 degrees from the line formed by connecting the vehicle to the closest point of the obstacle. At this point, the vehicle again checks its sensors to determine if the obstacle is still within its +- 45 degree sensor swath emanating from the front of the vehicle. If the vehicle still senses the obstacle after the first left turn, it will again swing left 50 degrees beyond the shortest line from the vehicle to the obstacle from the second observation. If the obstacle is not detected after any of the left clearing turns, the vehicle drives forward for one second, stops, and then turns right 90 degrees. The purpose of the 90-degree turn is to keep the vehicle close enough to the obstacle to continue to navigate around its perimeter. After the 90-degree turn to the right, the vehicle repeats the same sequence of sensor observations and left turns until the vehicle manages to move forward again and complete the next 90-degree right turn. If the vehicle does not sense the obstacle after the 90-degree turn, then it drives forward on that heading for one second, stops, and evaluates again for the presence of the obstacle. Essentially, the vehicle moves forward only when it senses no obstacle, and when it encounters an obstacle, conducts a series of left turns until it is able to navigate around the edge of the obstacle. When the vehicle reaches approximately the area where it originated the circumnavigation, it calculates the diameter of the obstacle based on a dead-reckoning navigation scheme, reports the obstacle's position and size, and makes a 90-degree turn to the left and drives off in the "reconnaissance mode" conducting random turns and driving at search speed. Figure 3.7 shows a trace of the autonomous vehicle from the surf-zone simulation performing this type of obstacle avoidance behavior, while Figure 3.8 shows a State-Logic Diagram for the obstacle mapping routine for the surf zone vehicle.

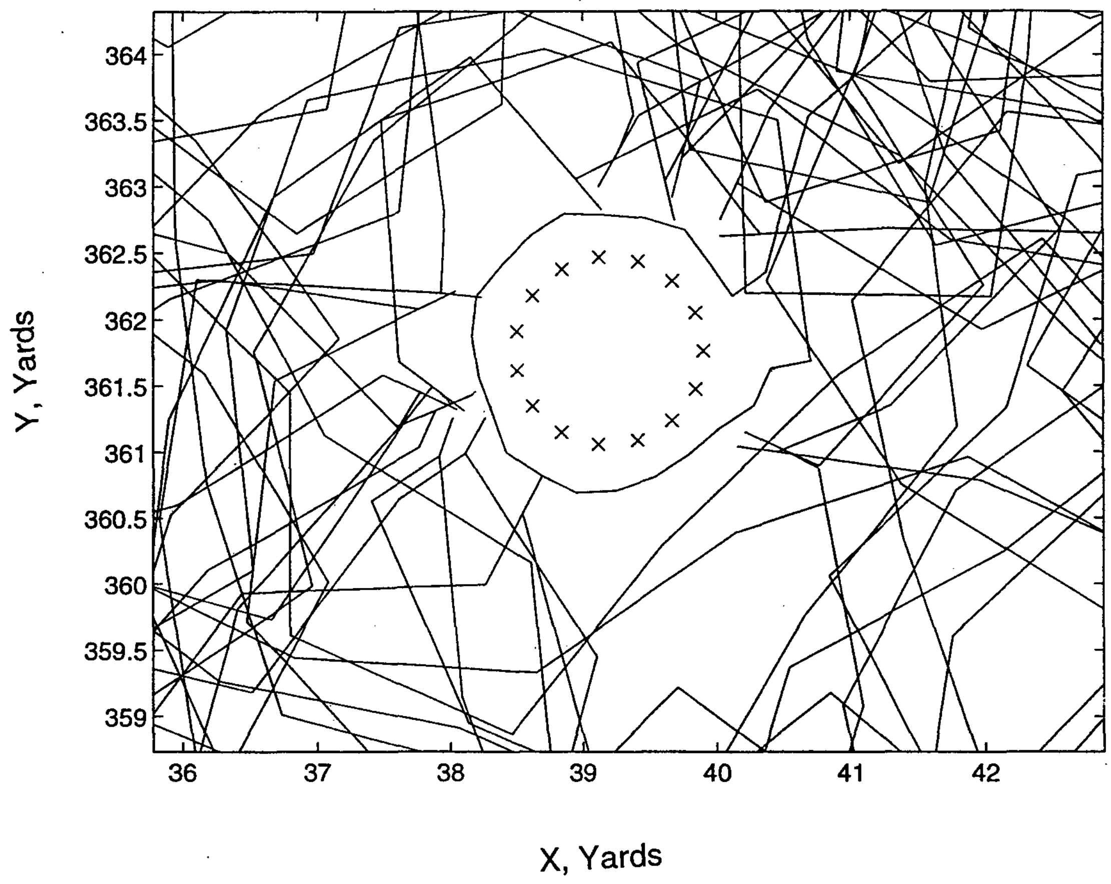


Figure 3.7 Surf-Zone Simulation State-Based Obstacle Mapping Trace

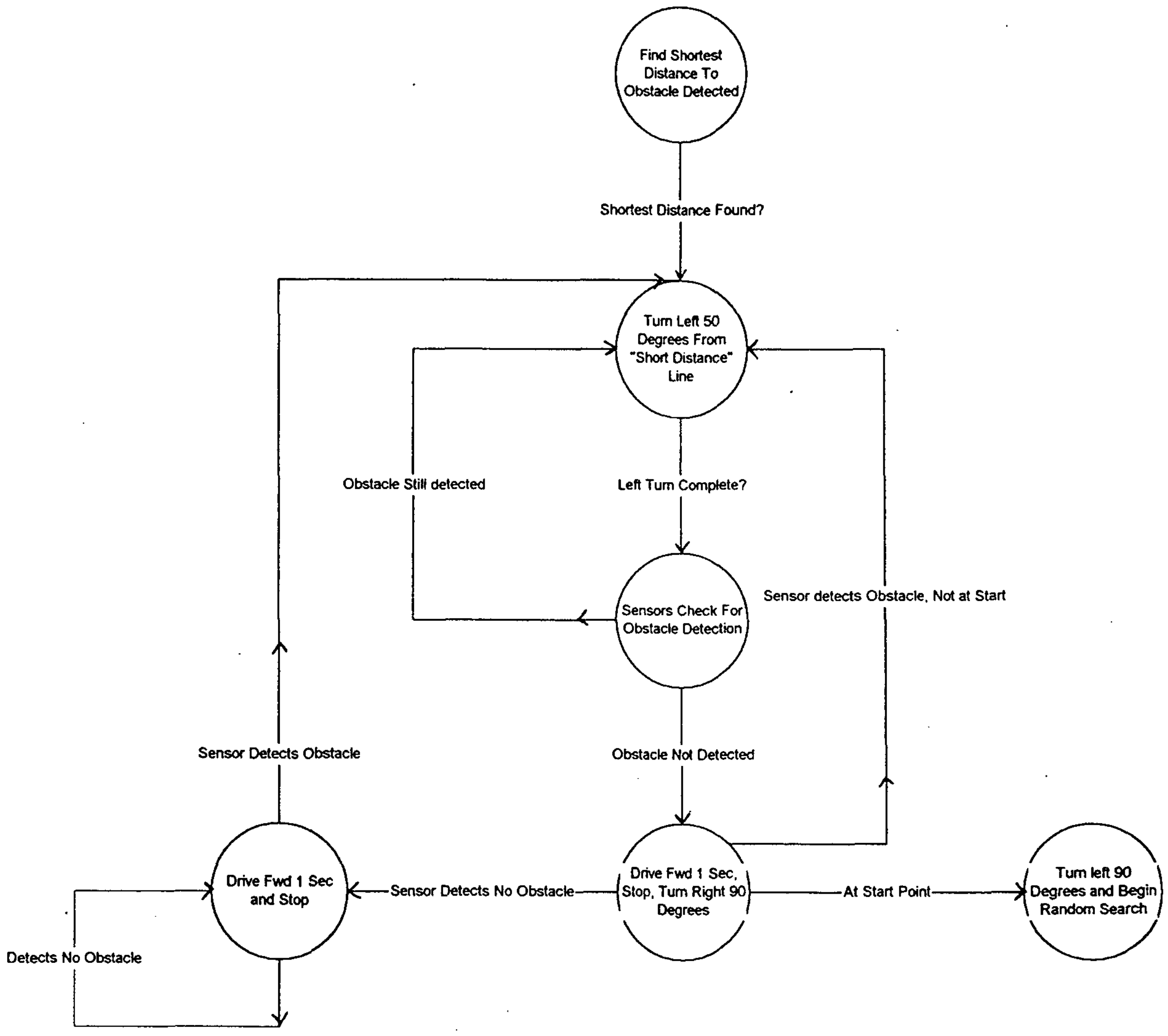

Find Shortest Distance To Obstacle Detected
Shortest Distance Found?
Turn Left 50 Degrees From "Short Distance" Line
Obstacle Still detected
Left Turn Complete?
Sensors Check For Obstacle Detection
Sensor detects Obstacle, Not at Start
Sensor Detects Obstacle
Obstacle Not Detected
Drive Fwd 1 Sec and Stop
Sensor Detects No Obstacle
Drive Fwd 1 Sec, Stop, Turn Right 90 Degrees
At Start Point
Turn left 90 Degrees and Begin Random Search
Detects No Obstacle

Figure 3.8 State Diagram For Surf Zone Obstacle Mapping Routine

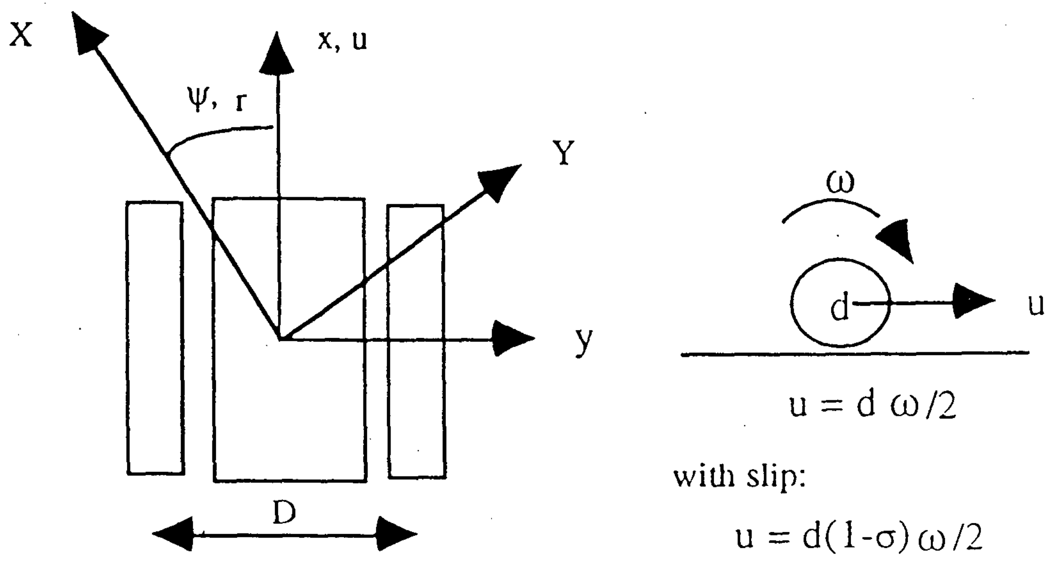
IV. VEHICLE DYNAMICS

A. GENERAL

The tracked vehicles that are simulated in the land-based and surf-zone models are classified as "tracked" vehicles as far as their locomotion and maneuverability is concerned. This is essentially the case with the Foster Miller and ISR vehicles, as well as the Lemming vehicle, which is quite similar to the Foster Miller variant, with the twin tracks and the ability to be symmetrical, that is, flip over and continue to drive. Also, a Navy BUG has been designed with four wheels for traction with differential wheel speed as the steering mechanism. Although the numerical simulations don't take into account the three-dimensional nature of a tracked vehicle moving over terrain or along the surf-zone bottom, the details of the vehicle dynamics do allow the simulations to contain the parameters that most clearly approximates the dynamic performance of the vehicle. Additionally, each vehicle uses a different sensor set and navigation method.

B. VEHICLE MODELING

Tracked vehicle control is essentially accomplished by differential rotation between right and left track rather than wheel turning as shown in Figure 4.1. The dynamic control of speed and heading in the simulation is continuous and is performed by sending command signals to the



$$u = d\,\omega/2$$

with slip:

$$u = d(1-\sigma)\,\omega/2$$

$0<\sigma<1$: slip factor

Figure 1

Forward Speed ,u and Rotational Rate, r are Related to the Average and the Differential Track Speeds

Figure 4.1 Tracked Vehicle Control

vehicle drive motors. A continuous control used in the simulations uses inverse kinematics and a control law that commands a heading rate proportional to the heading error. Thus, the guidance for position and control for heading is represented by the equations

$$r_{com} = K(\psi_{com}(t) - \psi(t))$$
$$u_{com} = \begin{Bmatrix} u_{max} & \text{in transit} \\ u_{search} & \text{in search} \end{Bmatrix}$$
$$\psi_{com}(t) = tan^{-1}\frac{X_k - X(t)}{Y_k - Y(t)} + \psi_{oa}(t)\ ;$$
where $(X_k,\ Y_k)$ *is the coordinate of the next target*

While the right / left motor speeds are derived from
$$\omega_l(t) = 2u_{com} / d + Dr_{com} / d;$$
$$\omega_r(t) = 2u_{com} / d + Dr_{com} / d;$$

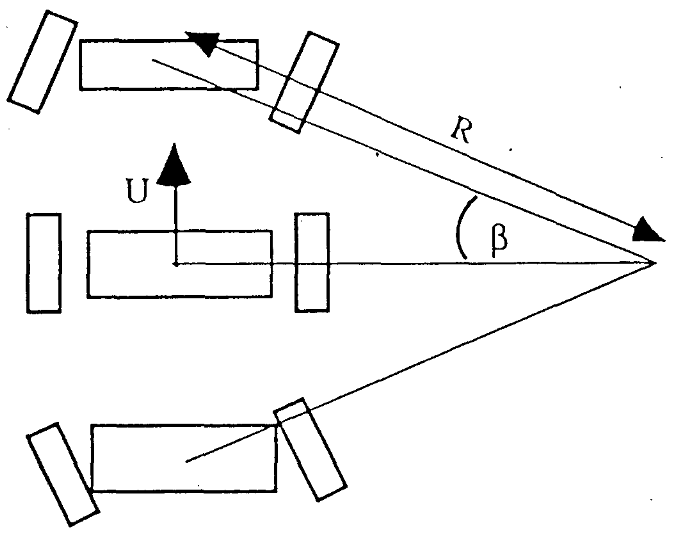
In this case, an added heading command is included that accounts for the heading command for randomization or that arising from the obstacle avoidance behavior when active. Therefore, the path of the vehicle is the forward solution of the model, and can be given by

$$u(kdt) = 0.5 * (\omega_l(kdt) + \omega_r(kdt))d$$
$$r(kdt) = (\omega_l(kdt) - \omega_r(kdt))d / D$$
$$\psi((k+1)dt) = \psi(kdt) + r(kdt) * dt$$
$$X((k+1)dt) = X(kdt) + u(kdt)\cos(\psi(kdt))$$
$$Y((k+1)dt) = Y(kdt) + u(kdt)\sin(\psi(kdt))$$

where d is the half-diameter of the wheels, and D is the track separation distance. As far as the issue of the precision of the control of heading and position, the precision is only as good as the precision of the sensor. In the model and simulation, relatively precise or poor heading control is modeled by an additive random bias to the compass output as it is used in the control command computation. Similarly, in the simulation of way-point navigation, errors in the positioning system outputs (X and Y) are corrupted by additive random bias where the spectral characteristics of known DGPS errors are used as modified by considerations of control update rate.

Track slippage can be modeled in the simulation by introducing a fractional slip between the wheel rotational speed commanded and the actual speed produced at the wheel ground interface [6]. The effects of track slippage are studied and seen to increase the effective navigational errors in acquiring a known location, although with an effective line-of-sight guidance law, and an accurate DGPS positioning system, the bugs are able to home to the target and transition into the next phase of the mission without difficulty. The effects of relatively large slippage have been studied and presented in [6]. For the steered vehicles, the kinematics of the steering motion arise differently and are modeled by a turn rate that is proportional to the forward speed and the wheel steering angle as shown in Figure 4.2.

Draper Vehicle Steering Geometry



$r = U / R$; $R = L / 2 / \tan\ \beta$

- β = steering angle
- R = radius of curvature, meters
- U = forward speed, m/sec.
- r = turn rate rad/sec.

Figure 4.2 Steered Vehicle Control

V. SIMULATION RESULTS

A. GENERAL

The simulations conducted in this thesis center around the modeling for the Surf-Zone Reconnaissance Mission, with some results from the EOD PUCA land-based scenarios presented, as well, to provide insight and transferable "lessons learned" to the simulations for the surf-zone model. Many of the findings of the EOD land-based scenario's numerical simulations were directly applicable to the production of data in the surf-zone model. The goals of these non-linear, Monte Carlo numerical simulations was to examine the search modeling in order to determine a proper search strategy for the Surf Zone Reconnaissance Mission. The simulations attempt to find a balance between total search time, sensing radius for the vehicle's sensors, time between heading changes, as well as vehicle speeds and quantities of vehicles, and then fuse that information to a suitable search strategy. These results allow for the determination of which parameters have the greatest effects on the efficiency of the search.

The simulations are conducted using programs written in the language C. The idea was to simulate with a core program, such as the one utilized in the EOD PUCA and surf-zone models, and then modify it as necessary to provide the required parameters for the given simulation scenario being conducted using the object-based capabilities of C. The parameters that were varied not only included the vehicle characteristics, but also the layout of the search area targets, obstacles in the vehicle's search area, total mission search time and time between heading changes, as well as sensor ranges for target and obstacle detection. Additionally, the probability of a correct detection with the various sensors could be adjusted to fit the scenario being simulated, thus allowing for the reality of imperfect sensors. Parameter changes were carried out in the programming and the input files to avoid code modifications and recompiling the programs used to run the given simulation scenarios.

The assumptions used in both the land-based and the surf-zone simulations included perfect navigation, target detection conducted using a "cookie-cutter" search, no target information passed between the search vehicles, and the assumption that the vehicles reflect off the UXO field and minefield lane boundaries at the vehicle incidence angle to the boundary, itself. The results of each simulation were stored in output files, and this included the clearance rates of the targets in the scenario over the simulation time, as well as the coordinates of the vehicular movement in order to graph traces of the vehicles movement for purposes of analysis and verification of the various algorithms. In the case of the surf zone simulation, the additional information found in the files was the threat evaluations for each of the targets in the surf zone lane by the vehicles, the number of visits for each of the targets by the vehicles during threat identification and the number of obstacles identified over simulation time. Since the Surf Zone mission includes reconnaissance, the identification of obstacles includes their positions in the approach lane, as well as the size of the obstacle.

The analysis conducted in this chapter attempts to determine the optimum search time for the given surf-zone scenario, the optimum obstacle and target sensor range for the reconnaissance mission, and the best time between heading changes for the random search using the autonomous vehicles. Additionally, the choice of the random-heading range is examined, that is, the limits of the possible course change, in degrees, from the previous heading to the left or right. With this information, further refinement of the vehicles will be possible as they are constructed and tested, and this information will prove useful in virtual simulations of the vehicles, as well.

B. EOD PUCA SIMULATIONS

Preliminary work conducted by [1] and [3] has provided many useful lessons in autonomous search behaviors that were ultimately useful in the surf zone modeling and simulations. These land-based, PUCA operations, with random point-target and point-obstacle placement showed some critical relationships in the search parameters that ultimately proved useful in the approach lane reconnaissance mission. Some simulations were conducted that involved a fleet of five vehicles that were used to conduct the search and carry-away mission in a 30-meter square UXO field with the random placements of point targets and obstacles. The goal was to examine the effect of the random heading generator, or specifically, the result of a heading change that involved angles +- 90 degrees from the original heading of the vehicle, and heading changes +- 120 degrees also from the original heading of the vehicle. The idea was to compare the acute angle heading changes to the obtuse angle heading changes and determine which method provided the highest clearance rate, with the results analyzed graphically to determine the best method for PUCA operations. A cleared target was considered a target detected and carried to pile point in the center of the UXO field for later disposal. The selection range of the targets was 10, 15, 20, 25, 30 and 40 targets, with all scenarios containing 20 obstacles. The mission time was chosen at 3 hours. Tables 5.1 and 5.2 show a numerical tabular comparison of this EOD scenario with 5 vehicles and the full range of targets. Table 5.1 represents the +-120 degree heading-

Results For EOD Model, Five Vehicles, 1000 Iterations, 20 Obstacles, 180 Minutes

Targets	Avg Time To Clear (minutes)	Avg Cleared Targets
10	138.6	9.9
15	163.5	14.8
20	179.3	19.7
25	195.5	24.5
30	201.2	29.4
40	216.3	39.2

Table 5.1 Results In PUCA Scenario With Heading Changes Of +-120 Degrees

Results For EOD Model, Five Vehicles, 1000 Iterations, 20 Obstacles, 180 Minutes

Targets	Avg Time To Clear (minutes)	Avg Cleared Targets
10	125.8	9.9
15	142.6	14.8
20	158.2	19.8
25	175.1	24.7
30	181.6	29.6
40	194.7	39.5

Table 5.2 Results In PUCA Scenario With Heading Changes Of +-90 Degrees

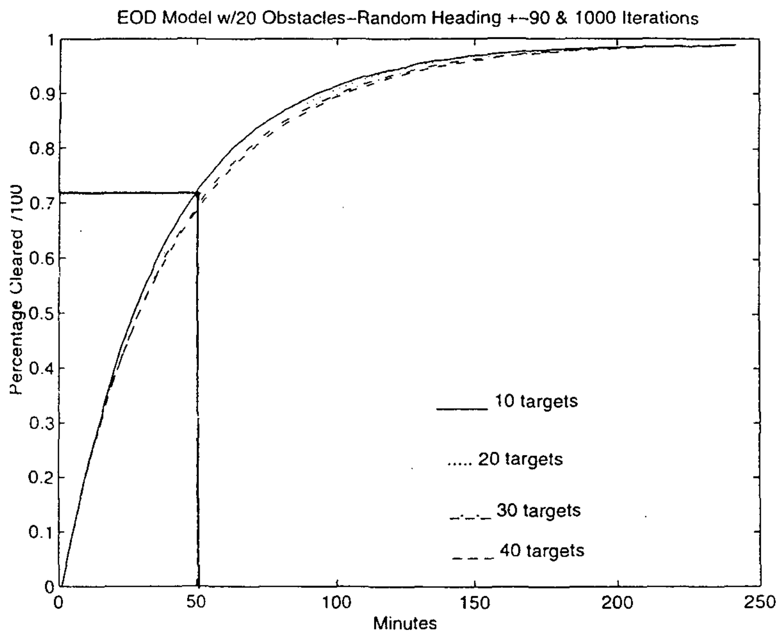


Figure 5.1 Target Clearance Results With Heading Changes Of +-90 Degrees

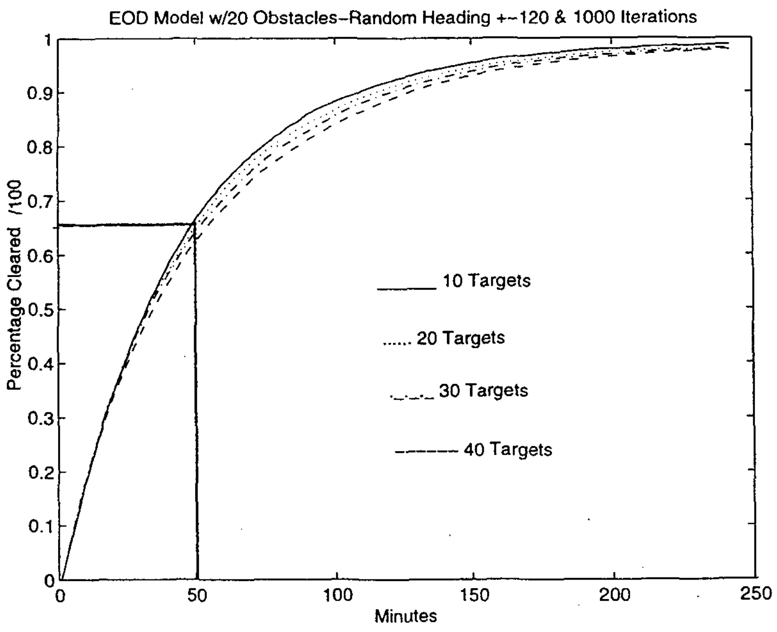


Figure 5.2 Target Clearance Results With Heading Changes Of +-120 Degrees

change model and Table 5.2 represents the +-90 degree heading-change model. The results indicate the improved clearance rate with the +-90 degree model. Figures 5.1 and 5.2 show graphically the comparison of the same performance figures, with the comparison focused just before the first hour of performance, or the 50-minute point. Again, the results indicate the acute-angle random- heading change generator algorithm produces a better percentage of UXO cleared over the given mission time. For the obtuse-angle heading changes, the 1000-run simulation average showed a 65 percent clearance of targets at the 50-minute point, and for the acute-angle heading changes over 1000 runs, a 72 percent clearance rate, also at the 50-minute point. Clearly, the use of a random-heading change of 90 degrees or less (to the left or right of original heading) is the preferred choice for the random search in the UXO field over the heading changes that are greater than 90 degrees. Figures 5.3 through 5.6 show the individual comparisons of the obtuse-angle and acute-angle heading-change models over the range of the targets tested in the 20-obstacle UXO field.

Another lesson learned in autonomous-vehicle robotic operations is a result of a comparison conducted in the land-based EOD PUCA scenario between different types of combinations of obstacle-avoidance algorithms. Specifically, the comparison was between the conventional boundary-reflection and state-based obstacle avoidance system, such as the one discussed in chapter three, with a state-based obstacle and state-based boundary avoidance system. Comparisons were sought regarding the effectiveness of this streamlined avoidance approach with the reflective boundary-avoidance and state-based obstacle-avoidance system discussed in chapter three for the EOD PUCA scenario, specifically the ability to clear the UXO field at a similar or better rate. If the clearance rates were comparable, or if the streamlined

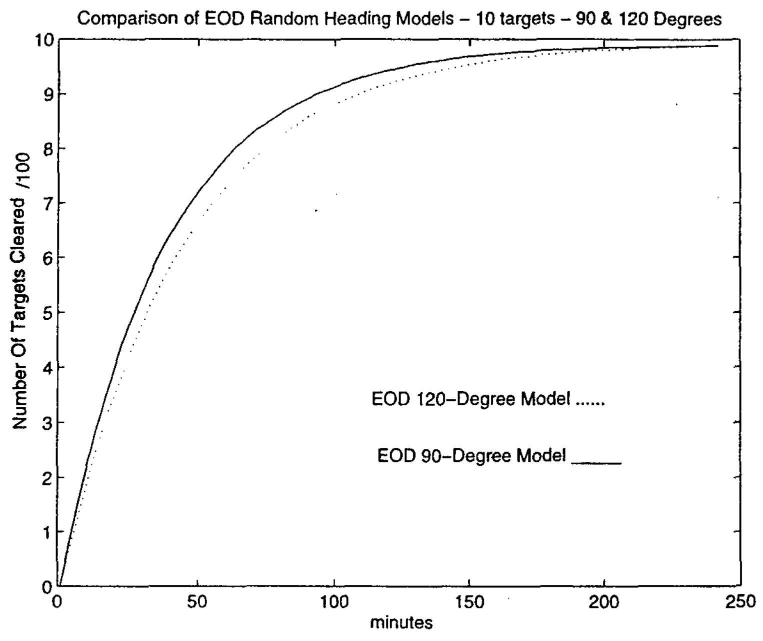


Figure 5.3 Comparisons, Clearance Results, Heading Changes of +-120 and +- 90 Degrees

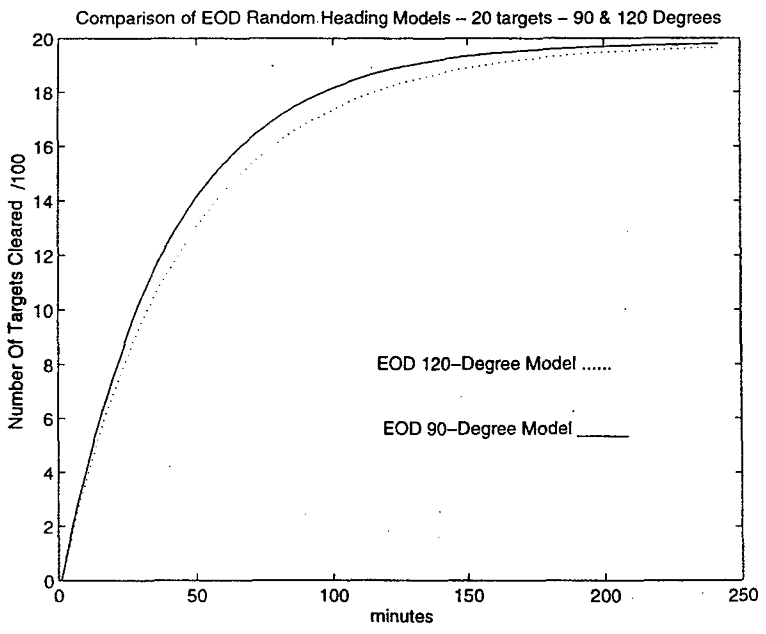


Figure 5.4 Comparisons, Clearance Results, Heading Changes of +-120 and +- 90 Degrees

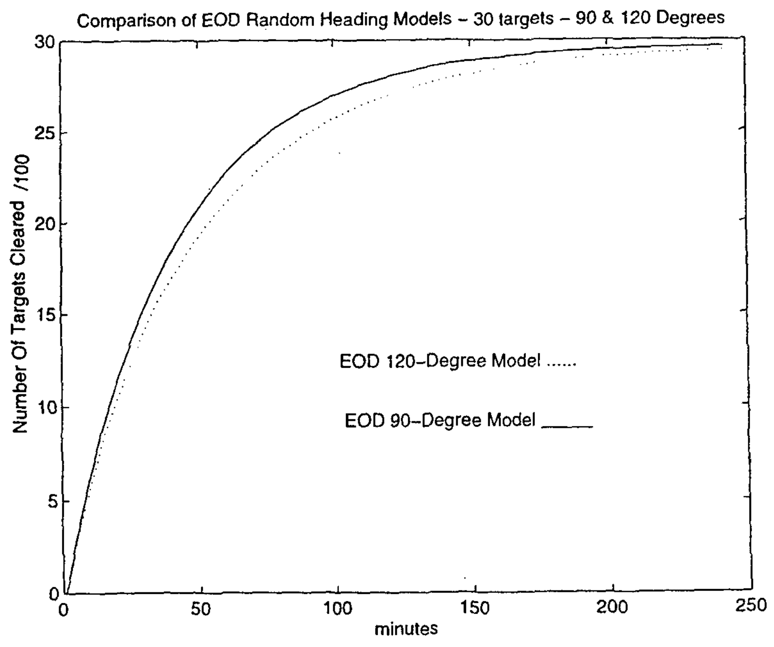


Figure 5.5 Comparisons, Clearance Results, Heading Changes of +-120 and +- 90 Degrees

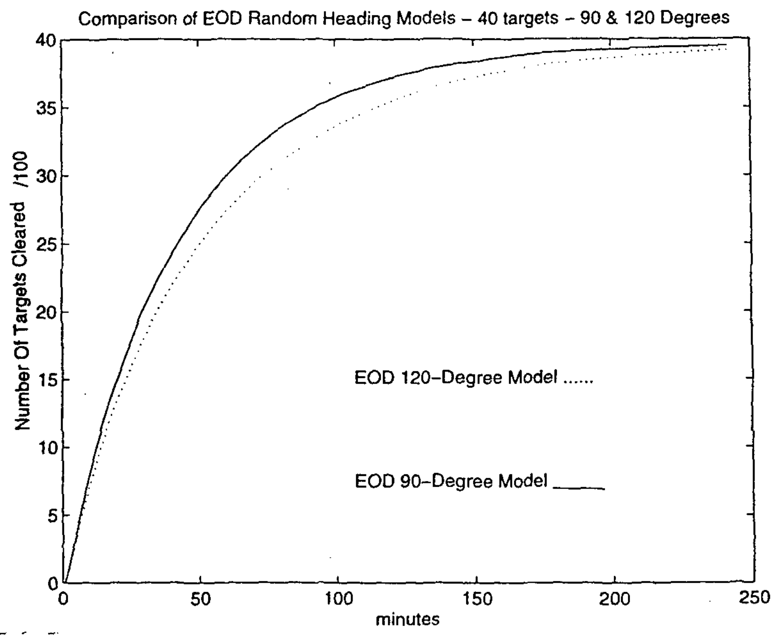


Figure 5.6 Comparisons, Clearance Results, Heading Changes of +-120 and +- 90 Degrees

obstacle-avoidance approach proved more effective, it would mean that a search vehicle could be constructed that would have a simpler and less costly design, whether the obstacle is the UXO field boundary, itself, or the obstacles in the UXO field.

A series of simulations were conducted to test this idea, with the UXO field chosen a 30-meter square site with 20 point obstacles and between 10 and 40 targets. The conventional boundary-reflection and state-based obstacle-avoidance system was called the Boundary-Reflection State-Based, or BRSB model, while the streamlined algorithm with State-Based boundary and State-Based obstacle avoidance was called the SBSB model. The mission time in the simulations was four hours and the target scenarios examined were 10, 20, 30 and 40 targets. As mentioned previously, there were 20 point obstacles in all the 1000-iteration simulation runs. Figures 5.7 and 5.8 show the results for the 10 and 20 target scenarios, while Figures 5.9 and 5.10 show the results for the 30 and 40 target scenarios. Clearly, in all cases, the streamlined avoidance algorithm was not able to produce a clearance rate comparable with the boundary reflection and state-based obstacle-avoidance system for mission times under four hours. The fact that the boundary-reflection avoidance routine puts the vehicle back in the field quickly, as well as avoiding the perimeter boundary, seems to be the deciding factor in this analysis of the different approaches.

C. SURF-ZONE RECONNAISSANCE SIMULATIONS

The goals for the Surf-Zone Reconnaissance Mission simulations were to determine an acceptable number of vehicles, within the constraints of the likely mission time and vehicle

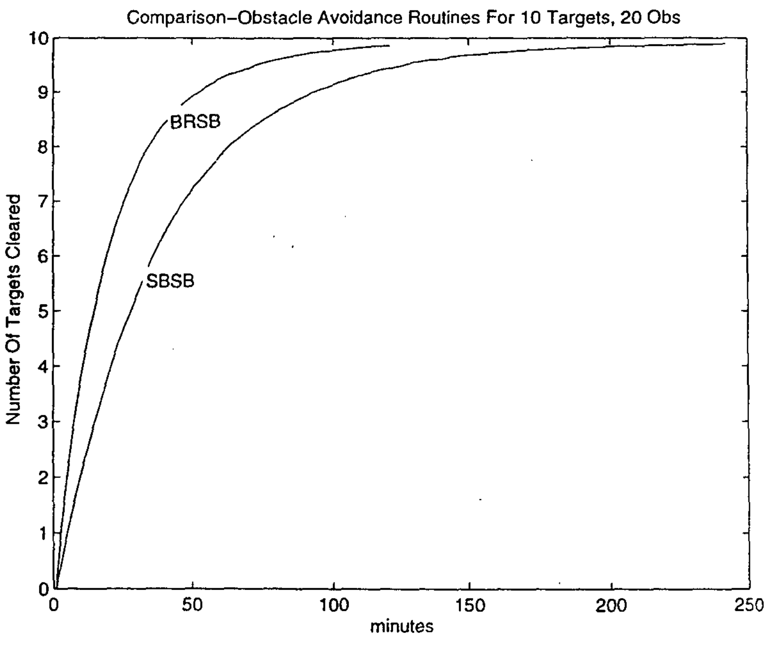


Figure 5.7 Comparisons, Clearance Results, BRSB and SBSB Boundary Avoidance

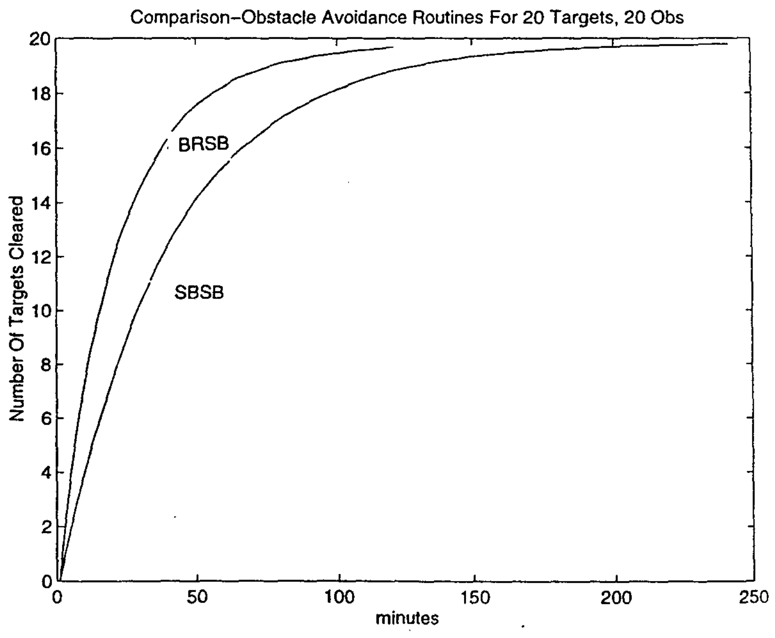


Figure 5.8 Comparisons, Clearance Results, BRSB and SBSB Boundary Avoidance

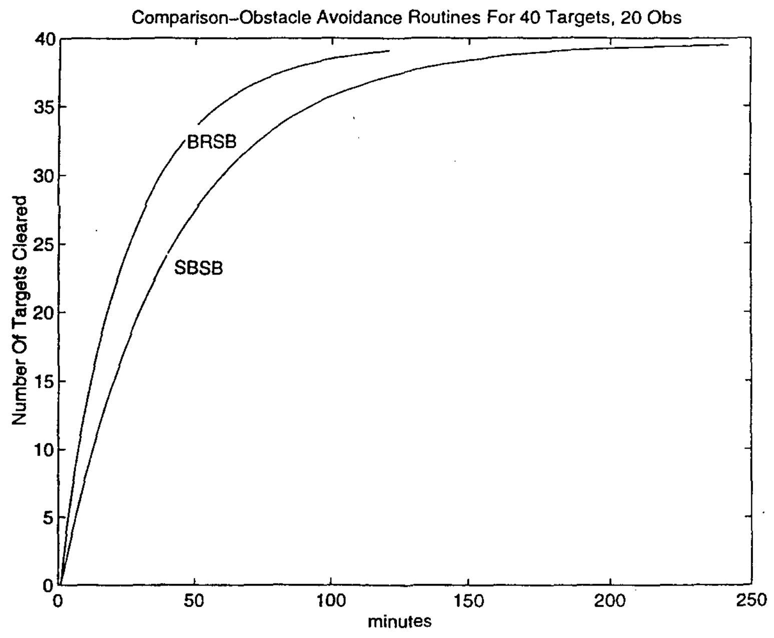


Figure 5.9 Comparisons, Clearance Results, BRSB and SBSB Boundary Avoidance

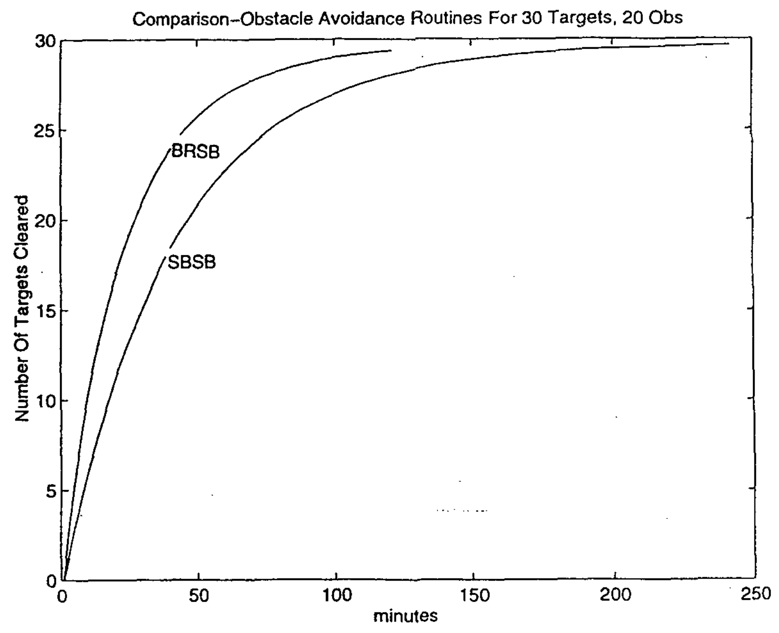


Figure 5.10 Comparisons, Clearance Results, BRSB and SBSB Boundary Avoidance

capabilities (4 hours), and to conduct the reconnaissance of the approach lane obstacles and identify all threat objects, such as mines, to a 95-percent simulated completion rate. Based on the likely energy available from the batteries, the initial studies and simulations focused around a four and six-hour mission, with the four-hour mission being the most likely. The simulation plan called for a matrix approach, that is, varying the number of vehicles while narrowing down suitable vehicle settings for random heading changes, the times between those random heading changes, and the feasible sensor ranges that would handle effectively the potentially layered targets and multiple-sized obstacles. For the given search- and obstacle-mapping algorithm discussed in Chapters II and III, the initial runs would be conducted without obstacles to test the ability of the code to run the full, 1000-iteration simulations in the approach lane with the 57 assigned targets discussed in Chapter II. With the initial simulations testing for identification rates of targets varying the heading changes and heading-change times, and examining the simulation runs for the number of visits to each of the respective targets by the vehicles in the subzones, the ability to assess the vehicles' performance could be completed over the set of 25, 50, 75 and 100 vehicle cases. Figure 5.11 shows a simulation trace of the deployment of the 25-vehicle scenario with the 57 targets in the approach lane. Additionally, the probability of detection was set at p=0.6 for this initial testing, with this probability of detection to be raised to .8 over the course of the full range of simulations.

The simulations of the surf-zone mission would also test the results learned from the EOD land-based scenarios, specifically the improved clearance times using the random-heading changes between +- 90 degrees and the fact that improved clearance rates were observed for longer time

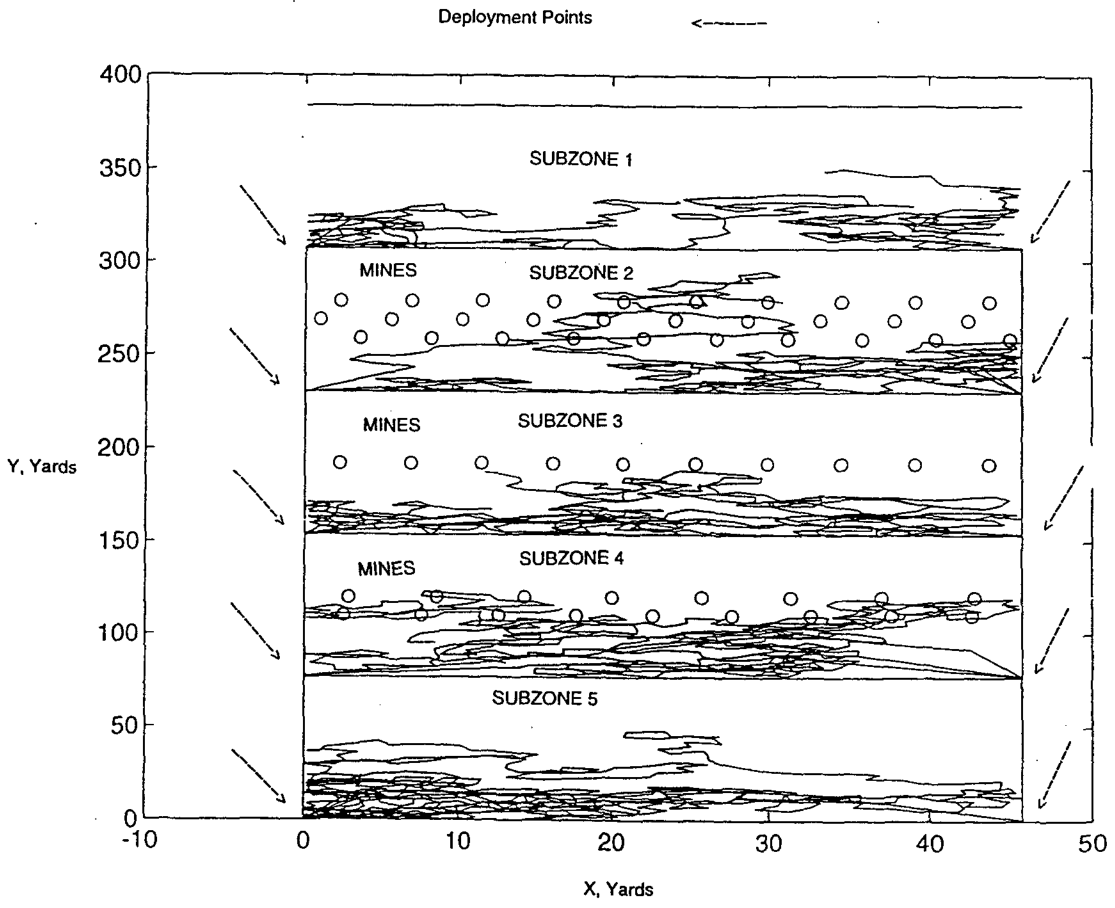


Figure 5.11 Simulation Trace - Deployment of 25 Vehicles in Approach Lane Subzones

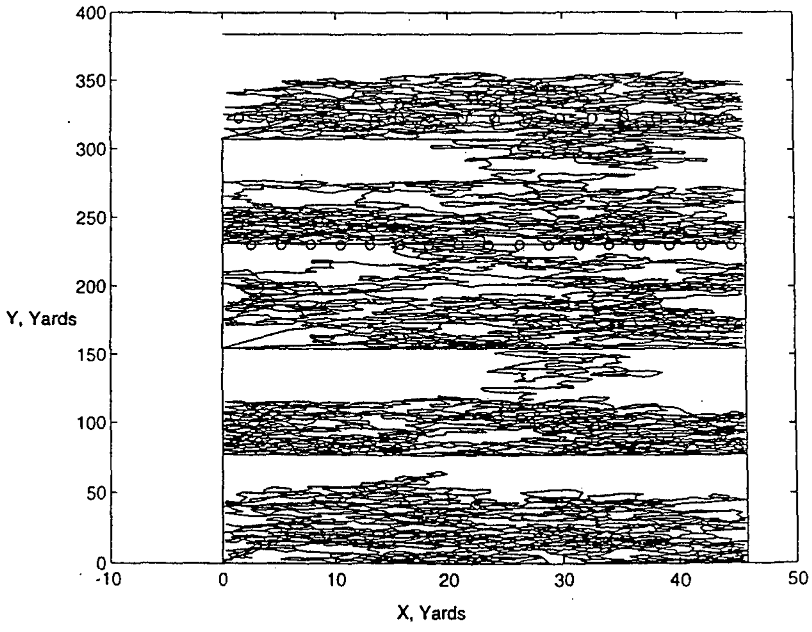


Figure 5.12 Trace of the 25-vehicle Scenario, Random Heading Changes +-120 Degrees

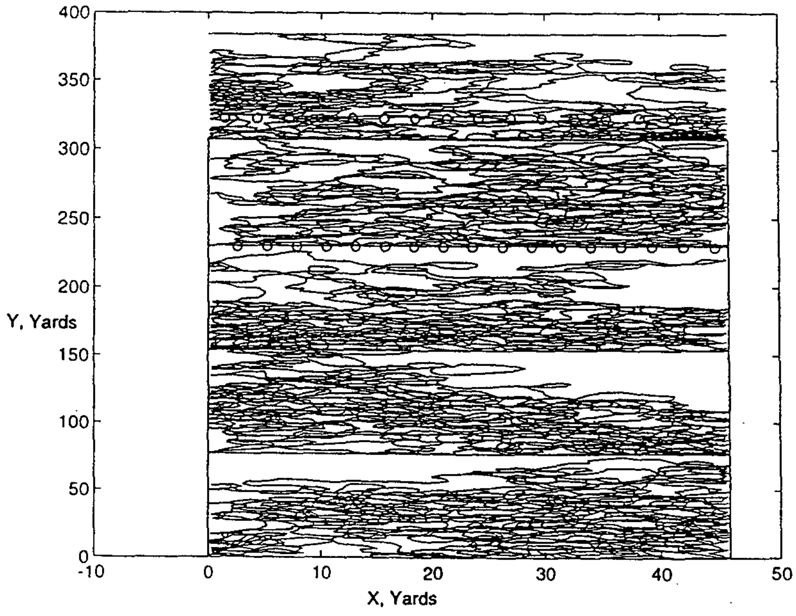


Figure 5.13 Trace of the 25-vehicle Scenario, Random Heading Changes +-90 Degrees

periods between heading changes. The objective would be to find the best range of heading changes, and the times between these heading changes that would bring the best results for the given parameters of the vehicle and the geometry of the approach lane. For the no-obstacle simulations, the time between heading changes was varied between 5, 7 and 9 seconds, and improved target identification rates were observed as the time was increased. Additionally, the use of heading changes of +- 90 degrees gave improved results than the choice of heading changes between +-120 degrees, just as in the land-based PUCA scenario. Figures 5.12 and 5.13 show the results of a simulation trace of 25 vehicles and the improved coverage of the scenario with the +-90 degree random-heading change over the +-120 degree random-heading change. A combination of long times between heading changes, and heading changes of +-90 degrees, not surprisingly, produced the best results. The results also indicate the possibility of completing a 4-hour mission with 25 vehicles while approaching an identification rate of 95 percent for the approach lane targets. The results for the no-obstacle simulations are summarized in Tables 5.3 - 5.6 and Figures 5.14 -5.22. Assumptions for these initial studies include perfect navigation, vehicles reflecting at the incidence angle at zone boundaries, and target detection is with a "cookie-cutter" search pattern. No target identification information is passed between vehicles.

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 25 vehicle Scenario:

	Avg Time	Average Identified TO
Random Heading - 120 degrees, 5 Seconds Between Heading Change.	239.32 min	50.47
Random Heading - 120 degrees, 7 Seconds Between Heading Change.	239.27 min	53.24
Random Heading - 120 degrees, 9 Seconds Between Heading Change.	237.98 min	54.28
Random Heading - 90 degrees, 5 Seconds Between Heading Change.	239.40 min	53.21
Random Heading - 90 degrees, 7 Seconds Between Heading Change.	237.77 min	54.58
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	236.65min	55.01

Table 5.3 Summary of Target Identification Results-25 vehicles

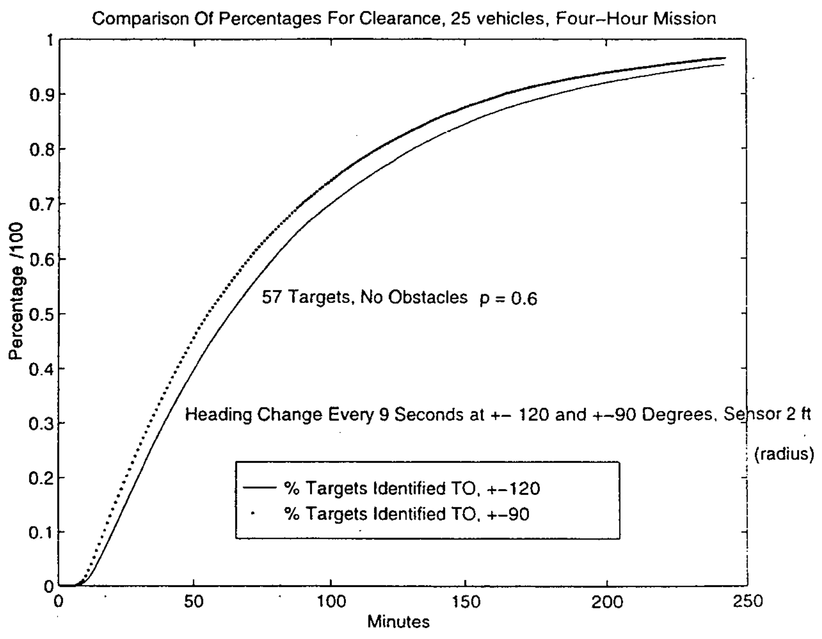


Figure 5.14 25 Vehicles, Identification Rate Comparison, Heading Changes

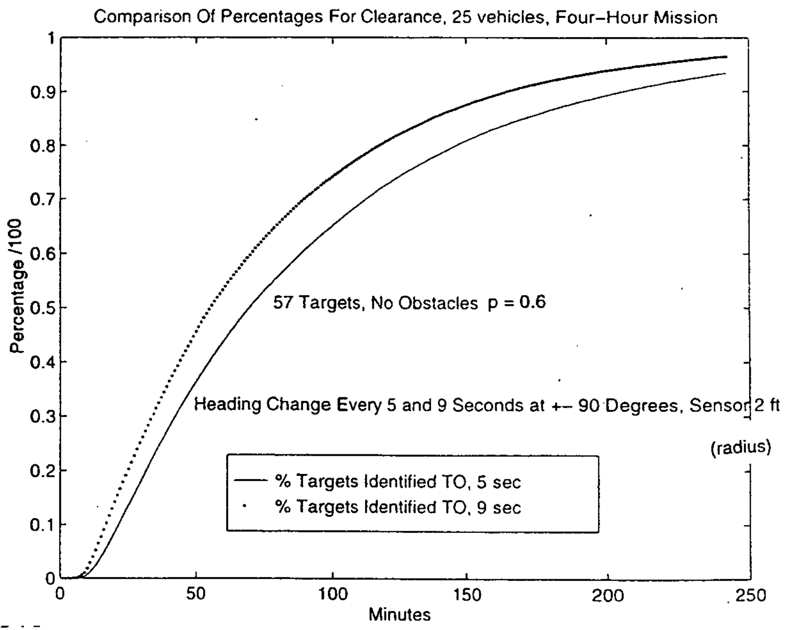


Figure 5.15 25 Vehicles, Identification Rate Comparison, Time Between Heading Changes

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 50 vehicle Scenario:

	Avg Time	Average Identified TO
Random Heading - 120 degrees, 5 Seconds Between Heading Change.	221.52 min	56.14
Random Heading - 120 degrees, 7 Seconds Between Heading Change.	195.57 min	56.70
Random Heading - 120 degrees, 9 Seconds Between Heading Change.	179.20 min	56.85
Random Heading - 90 degrees, 5 Seconds Between Heading Change.	195.18 min	56.76
Random Heading - 90 degrees, 7 Seconds Between Heading Change.	173.50 min	56.89
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	161.55 min	56.94

Table 5.4 Summary of Target Identification Results-50 vehicles

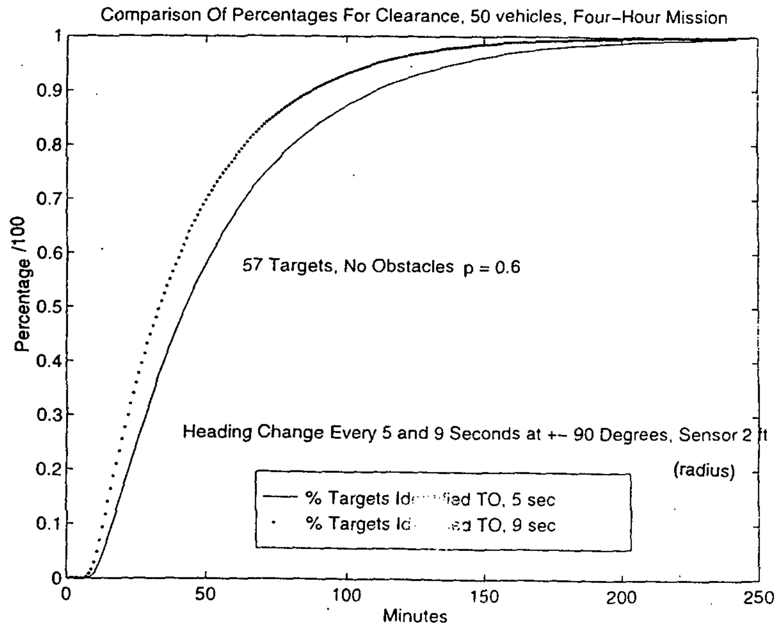


Figure 5.16 50 Vehicles, Identification Rate Comparison, Heading Changes

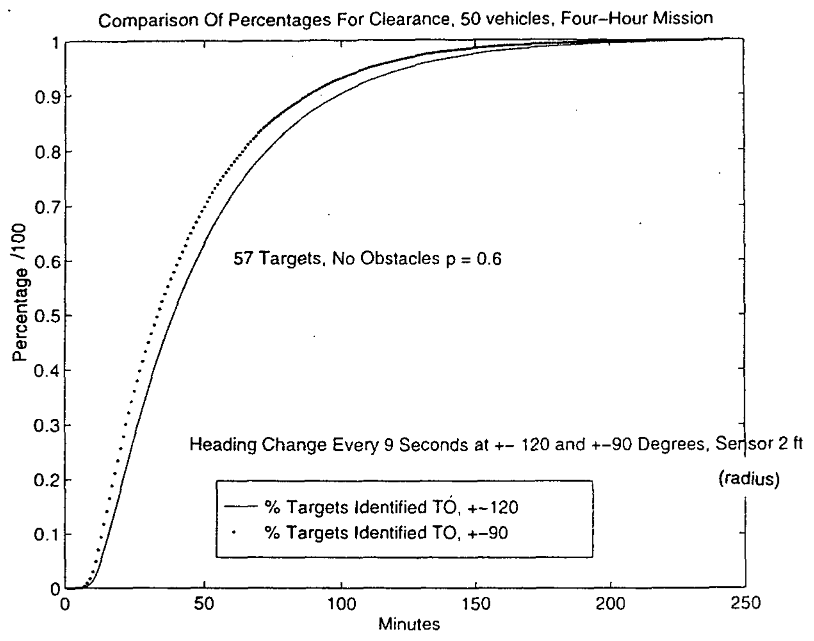


Figure 5.17 50 Vehicles, Identification Rate Comparison, Time Between Heading Changes

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 75 vehicle Scenario:

	Avg Time	Average Identified TO
Random Heading - 120 degrees, 5 Seconds Between Heading Change.	177.43 min	56.87
Random Heading - 120 degrees, 7 Seconds Between Heading Change.	145.33 min	56.97
Random Heading - 120 degrees, 9 Seconds Between Heading Change.	127.97 min	56.99
Random Heading - 90 degrees, 5 Seconds Between Heading Change.	140.18 min	56.98
Random Heading - 90 degrees, 7 Seconds Between Heading Change.	122.28 min	56.99
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	113.85min	56.998

Table 5.5 Summary of Target Identification Results-75 vehicles

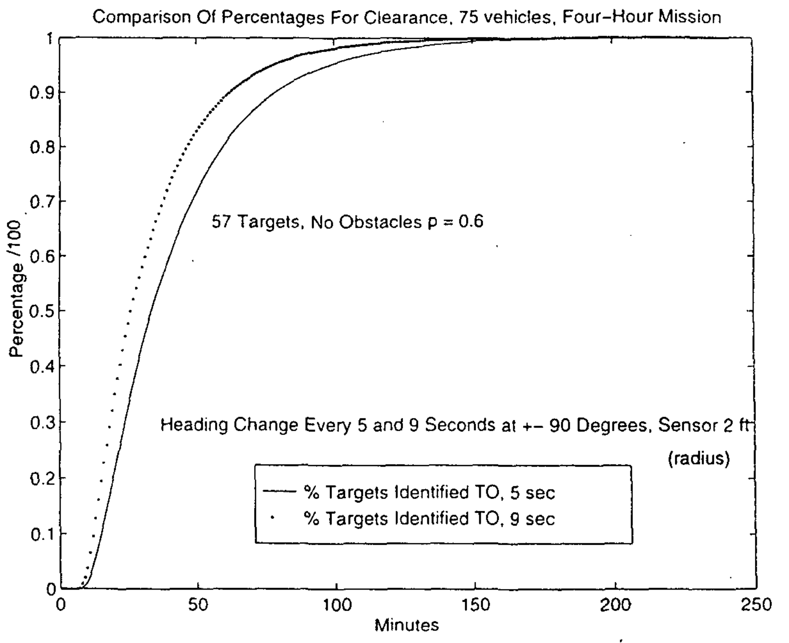


Figure 5.18 75 Vehicles, Identification Rate Comparison, Heading Changes

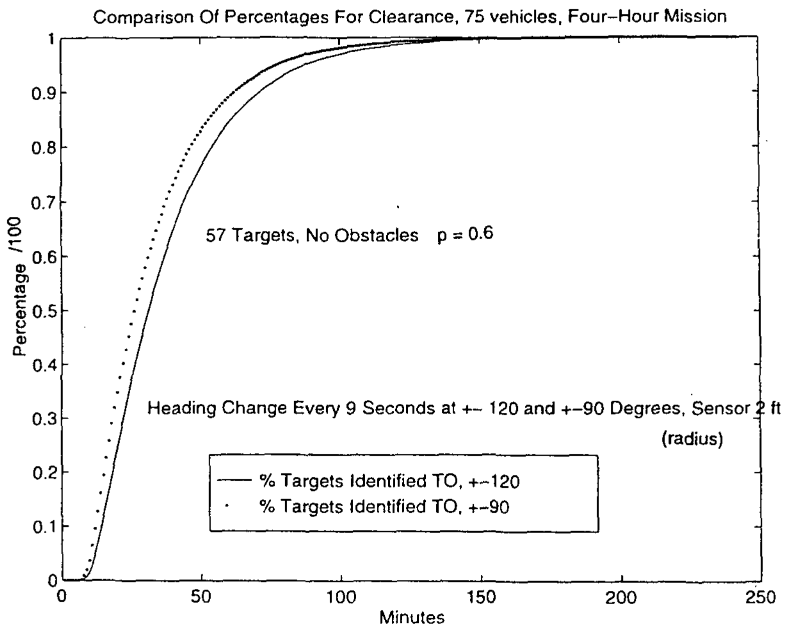


Figure 5.19 75 Vehicles, Identification Rate Comparison, Time Between Heading Changes

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 100 vehicle Scenario:

	Avg Time	Average Identified TO
Random Heading - 120 degrees, 5 Seconds Between Heading Change.	142.72 min	56.99
Random Heading - 120 degrees, 7 Seconds Between Heading Change.	114.40 min	56.995
Random Heading - 120 degrees, 9 Seconds Between Heading Change.	100.40 min	57.00
Random Heading - 90 degrees, 5 Seconds Between Heading Change.	113.0 min	56.997
Random Heading - 90 degrees, 7 Seconds Between Heading Change.	96.03 min	57.00
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	88.48 min	56.999

Table 5.6 Summary of Target Identification Results-100 vehicles

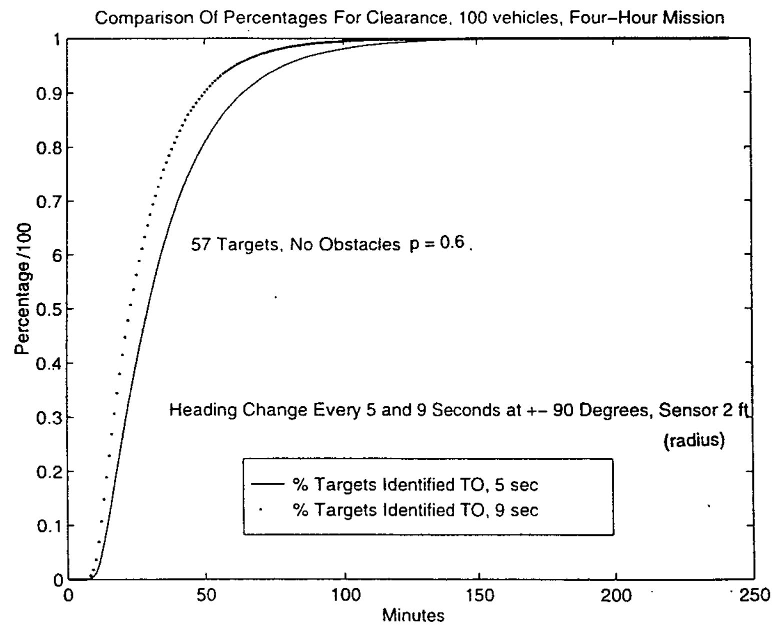


Figure 5.20 100 Vehicles, Identification Rate Comparison, Heading Changes

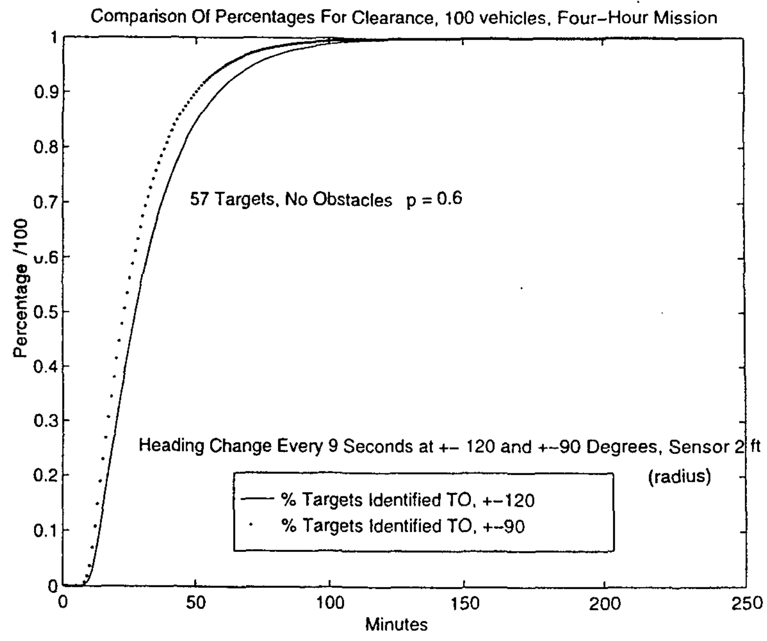


Figure 5.21 100 Vehicles, Identification Rate Comparison, Time Between Heading Changes

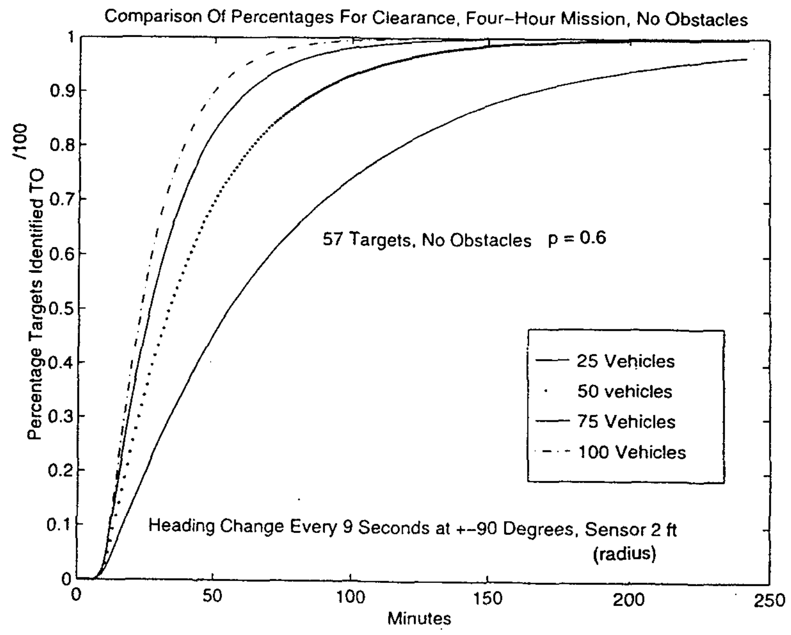


Figure 5.22 Summary Of Identification Rates, All Vehicle Groups, No Obstacles

A logical progression of the initial 1000-iteration runs of no-obstacle surf-reconnaissance simulations was the examination of the more-likely ASV fleet sizes of 25 and 50 vehicles with higher probabilities of detection, specifically with p=0.7 and p=0.8. Probability of detection values such as these could certainly be realized, and the simulations would show numerically what results could be expected for the ASV groups of 25 and 50 vehicles possessing these improved sensor detection capabilities. Even with the likelihood of a 4-hour mission, simulations of the 25 and 50 vehicle fleets with a 6-hour scenario were also conducted with the upgraded sensor detection values, with the intention of providing data to determine if significant results could be expected with the improved battery life and would the extended time be cost-efficient. Based on the initial data and confirmation of the better identification rates using heading changes of +-90 degrees and a time between heading changes of 9 seconds, these values were also incorporated into this second group of simulations. Again, the simulation assumptions for the approach lane scenario are the same and the vehicles are assumed to be completely autonomous. Sensor range for this second group of simulations was set at a 2 feet radius. The results are shown in Tables 5.7 - 5.10 and Figures 5.23 -5.26.

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 25 vehicle Scenario With 4-Hour Mission Time, p=0.7

Random Heading - 90 degrees,	Avg Time	Average Identified TO
9 Seconds Between Heading Change.	228.48 min	56.97

Results For 25 vehicle Scenario With 4-Hour Mission Time, p=0.8

Random Heading - 90 degrees,	Avg Time	Average Identified TO
9 Seconds Between Heading Change.	218.87 min	56.99

Table 5.7 Summary Of Results, 25 vehicles, 4-hour Mission

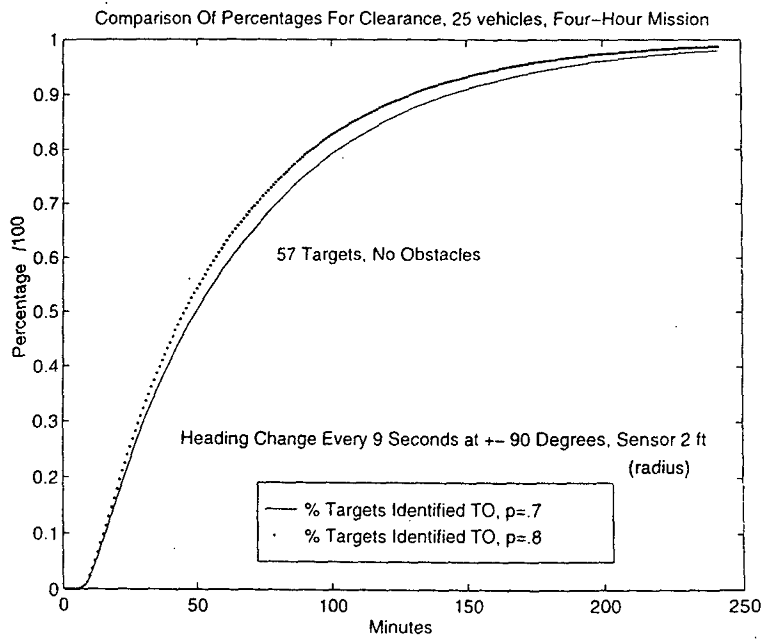


Figure 5.23 Clearance Rates, 25 vehicles, 4-Hour Mission, p=0.7 and 0.8, No Obstacles

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 25 vehicle Scenario With 6-Hour Mission Time, p=0.7

	Avg Time	Average Identified TO
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	277.2 min	56.79

Results For 25 vehicle Scenario With 6-Hour Mission Time, p=0.8

	Avg Time	Average Identified TO
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	248.25 min	56.90

Table 5.8 Summary Of Results, 25 vehicles, 6-hour Mission

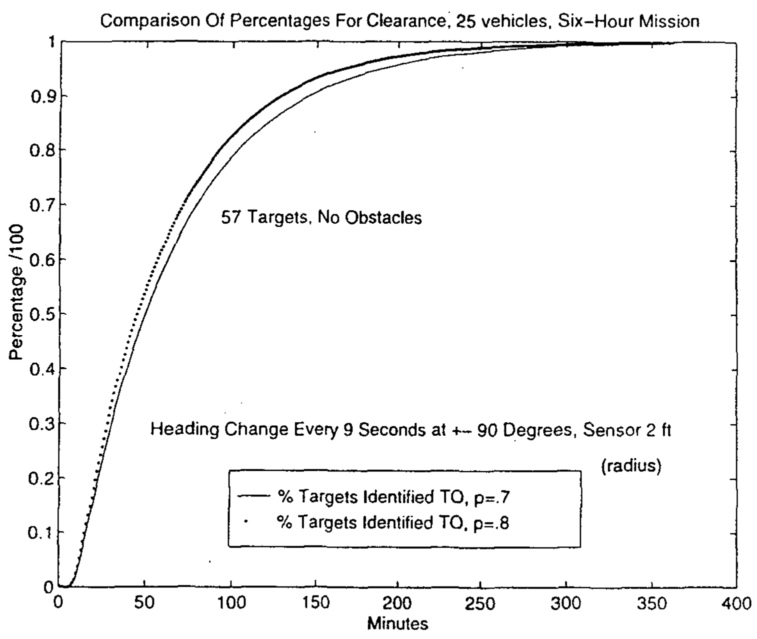


Figure 5.24 Clearance Rates, 25 vehicles, 6-Hour Mission, p=0.7 and 0.8, No Obstacles

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 50 vehicle Scenario With 4-Hour Mission Time, p=0.7

	<u>Avg Time</u>	<u>Average Identified TO</u>
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	145.52 min	56.97

Results For 50 vehicle Scenario With 4-Hour Mission Time, p=0.8

	<u>Avg Time</u>	<u>Average Identified TO</u>
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	128.77 min	56.99

Table 5.9 Summary Of Results, 50 vehicles, 4-hour Mission

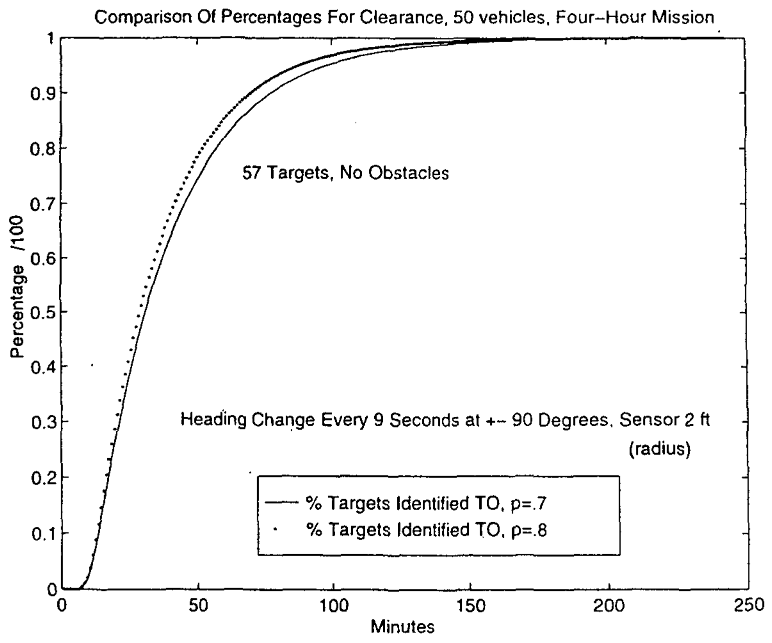


Figure 5.25 Clearance Rates, 50 vehicles, 4-Hour Mission, p=0.7 and 0.8, No Obstacles

Simulation Results - 1000 Iterations w/ 57 Targets and No Obstacles

Results For 50 vehicle Scenario With 6-Hour Mission Time, p=0.7

	Avg Time	Average Identified TO
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	145.83 min	56.99

Results For 50 vehicle Scenario With 6-Hour Mission Time, p=0.8

	Avg Time	Average Identified TO
Random Heading - 90 degrees, 9 Seconds Between Heading Change.	130.95 min	56.99

Table 5.10 Summary Of Results, 50 vehicles, 6-hour Mission

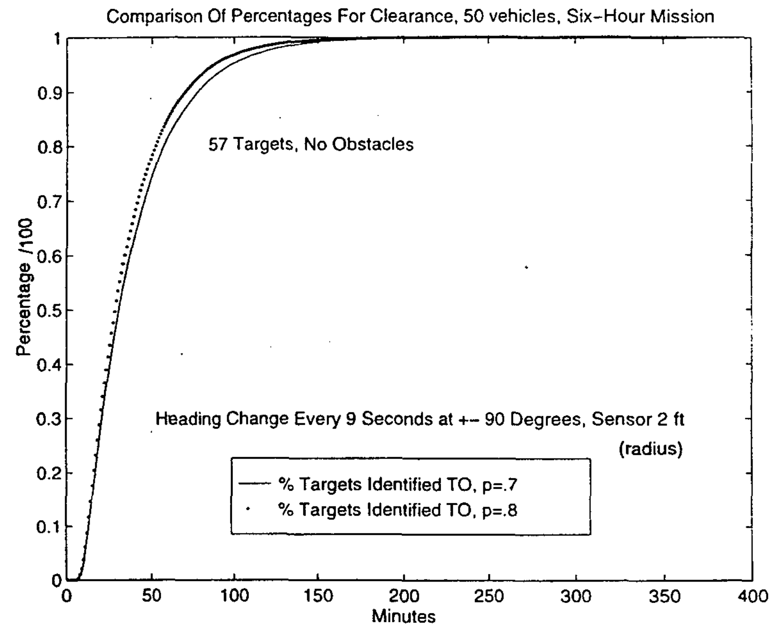


Figure 5.26 Clearance Rates, 50 vehicles, 6-Hour Mission, p=0.7 and 0.8, No Obstacles

1. Obstacles and Mapping

The objective of the last group of simulations was to take all the information and proven techniques of the previous simulations in the surf zone and apply them to a full target-identification and obstacle-mapping mission in the approach-lane scenario. The obstacles simulated in all the subzones were randomly-placed "rocks", and the rocks simulated were also randomly-sized from 1 to 10 feet. As in the previous simulations, the goal was to determine the identification times to be expected with the deployed vehicles, with the additional information studied being the mapping of the obstacles over time as a percentage of the total obstacles placed in the approach lane. For the purposes of this last group of simulations, it was decided to place 50 random obstacles in the lane subzones, and with that parameter fixed, find a time between heading changes that would maximize the mapping and identification times and confirm the faster rates in the target and obstacle environment typically associated with heading changes of +-90 degrees. Additionally, the results from the target and obstacle environment would be compared with "no-obstacle" runs with the same heading-change parameters to determine any relationships associated with the addition of obstacles in regards to identification times. The size of the fleet simulated was 25 vehicles, and for this last group of simulations, information on the obstacles mapped is passed between the vehicles in order to prevent any redundant mapping of obstacles in the respective subzones. This would allow us to see if this type of cooperative behavior can allow the identification and reconnaissance goals for the surf-zone obstacles and targets to be completed within the 4-hour mission. All of the simulations in the last group were 4-hour missions with 40 simulation iterations per mission. Figure 5.27 shows the simulation trace of the approach lane with the placement of the 57 targets and the 50 randomly-sized and randomly positioned obstacles, while Figures 5.28 and 5.29 show in greater detail the way the rocks were simulated using groupings of point obstacles in the shape of rock-like objects.

As previously mentioned, it was very desirable, for the given speed of the vehicles and the geometry of the approach lane to determine a time between heading changes (TBHC) that would optimize performance of identification and obstacle mapping. With the previous results from the first two groups of simulations indicating improved performance as the TBHC value was increased, a series of simulations were conducted slowly increasing the TBHC value until the identification and mapping rates slowly reached an acceptable value. This series of simulations was conducted using the heading changes of +-90 degrees. Figures 5.30-5.35 show the results of this series of simulations. TBHC values over 17 seconds seemed to indicate very little improvements in identification and mapping times, suggesting for the random search that there is a limiting TBHC value that will optimize your search for the given search area, and that a random choice of TBHC values is not recommended. This was the case for mapping obstacles and identifying targets.

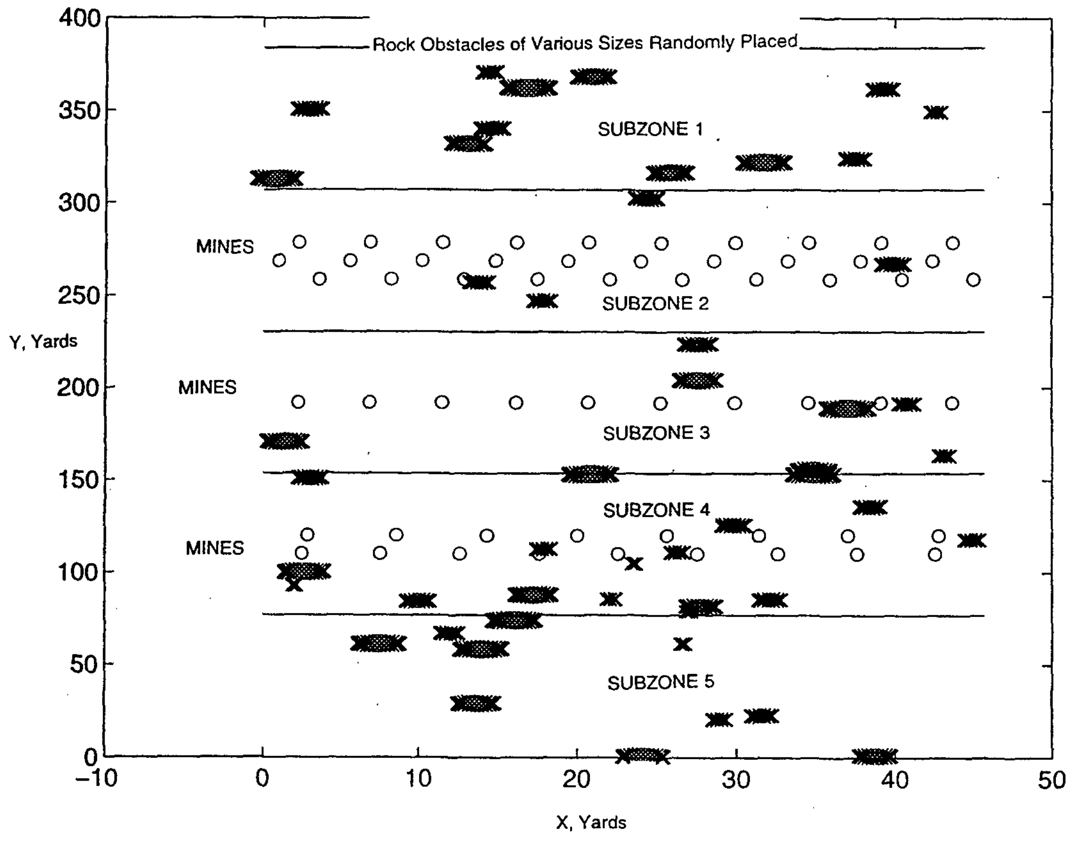


Figure 5.27 Simulation Trace, Approach Lane With Targets and Obstacles

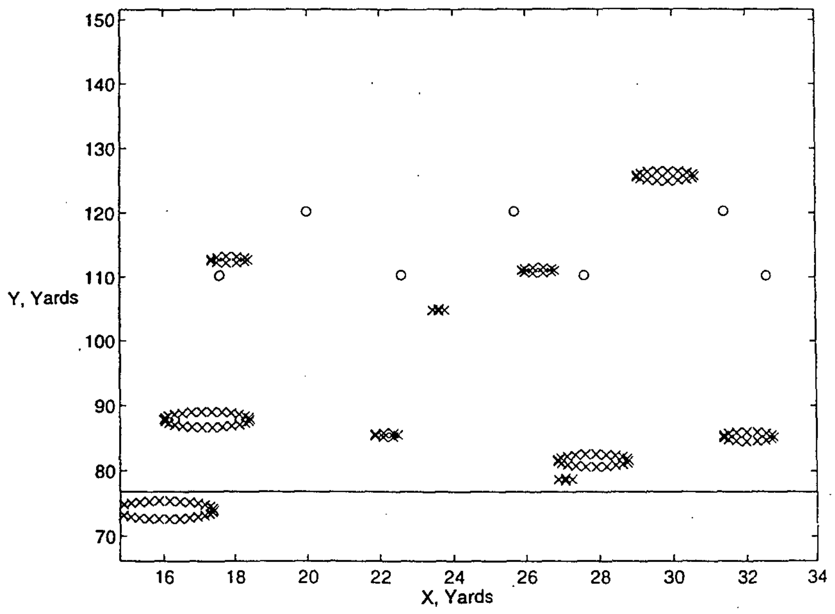


Figure 5.28 Simulation Trace, Approach Lane With Targets and Obstacles

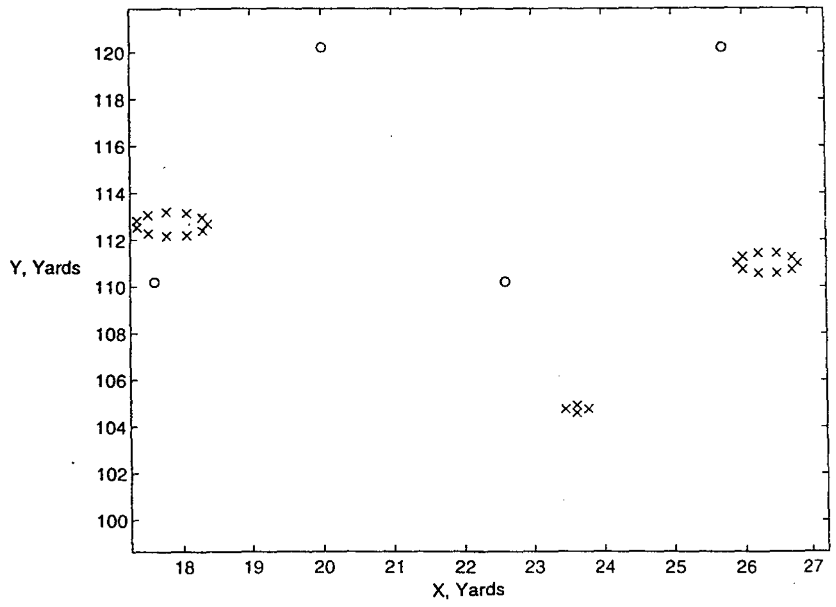


Figure 5.29 Simulation Trace, Approach Lane With Targets and Obstacles

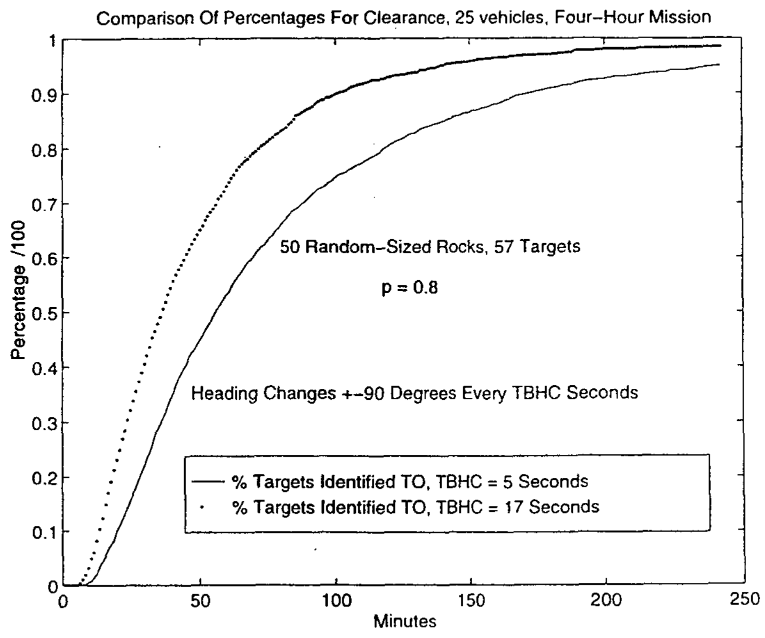


Figure 5.30 Comparison of Clearance, 25 vehicles With Varying TBHC, 50 obstacles

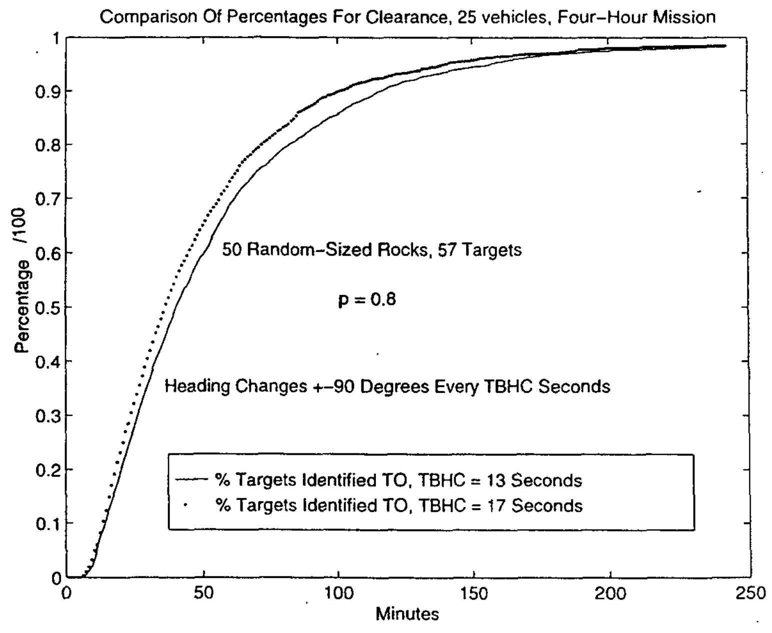


Figure 5.31 Comparison of Clearance, 25 vehicles With Varying TBHC, 50 obstacles

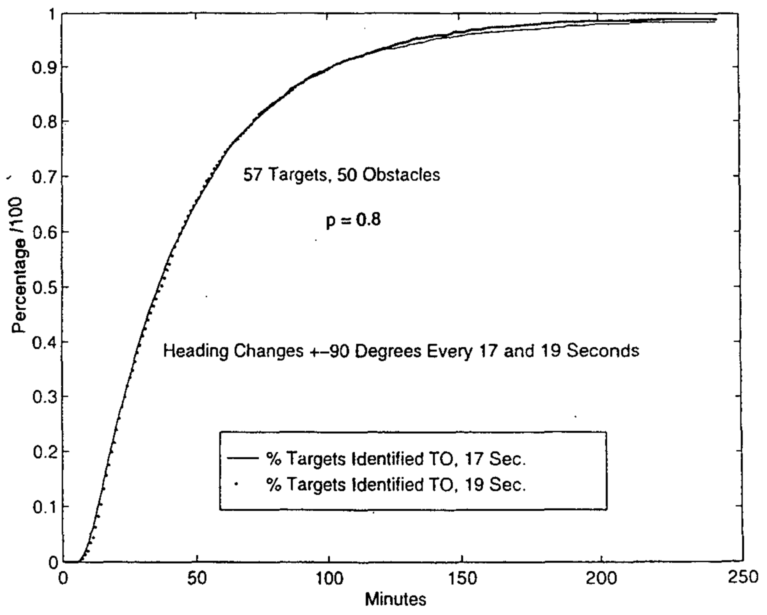


Figure 5.32 Comparison of Clearance, 25 vehicles With Varying TBHC, 50 obstacles

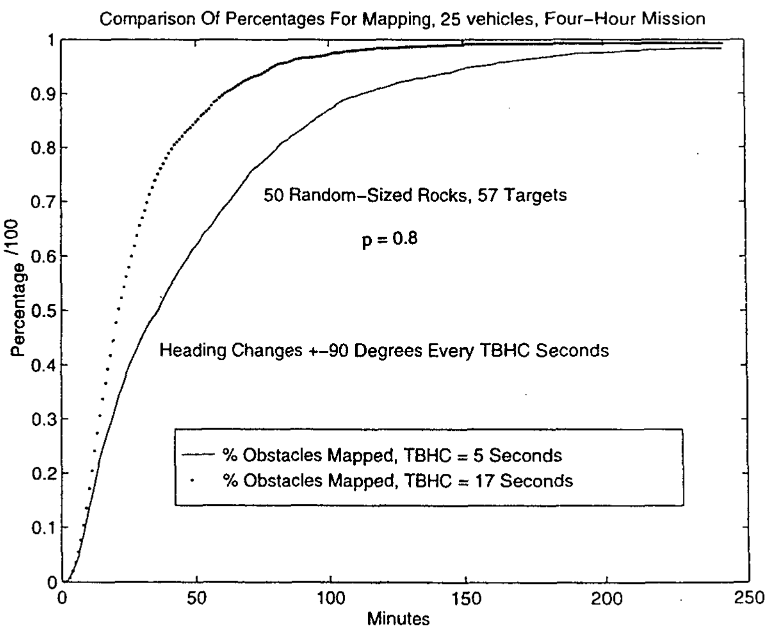


Figure 5.33 Comparison of Mapping, 25 vehicles With Varying TBHC, 50 obstacles

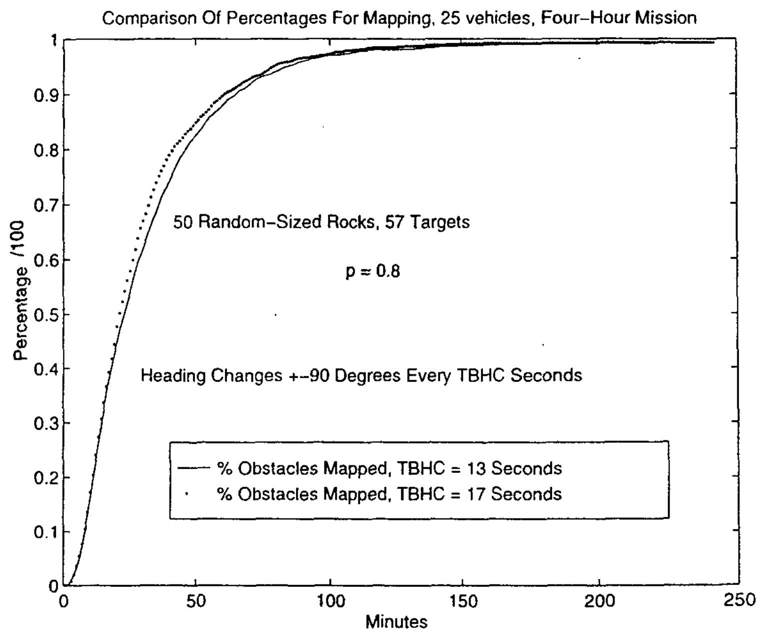


Figure 5.34 Comparison of Mapping, 25 vehicles With Varying TBHC, 50 obstacles

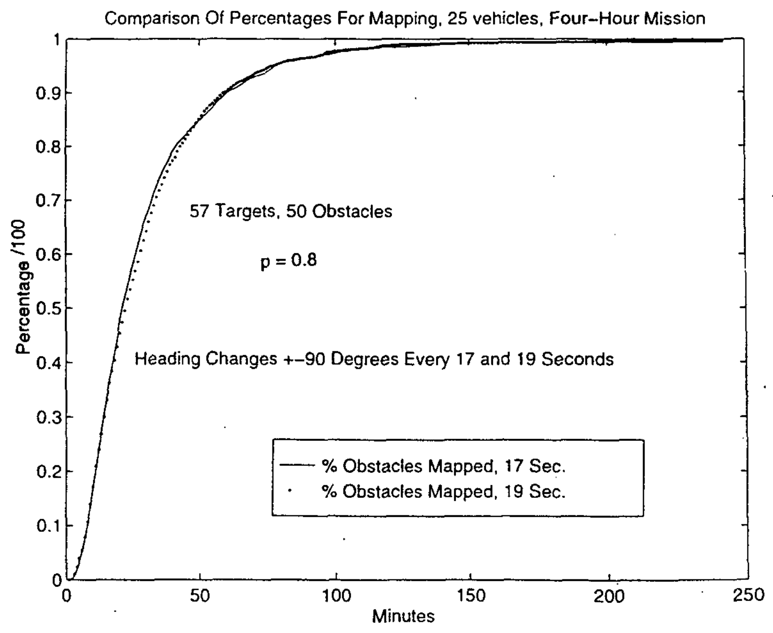


Figure 5.35 Comparison of Mapping, 25 vehicles With Varying TBHC, 50 obstacles

It was desirable to confirm the use of the choice of the uniformly-distributed random-heading change, therefore a series of simulations with the target and obstacle environment were conducted to compare the choice of heading changes of +-120 degrees and +-90 degrees. The choice of TBHC values utilized were 9, 13 and 17 seconds to check the results across the range of possible TBHC values. The simulation results showed improved identification rates for all TBHC values using the random-heading changes of +- 90 degrees. These results are shown in Figures 5.36 - 5.38.

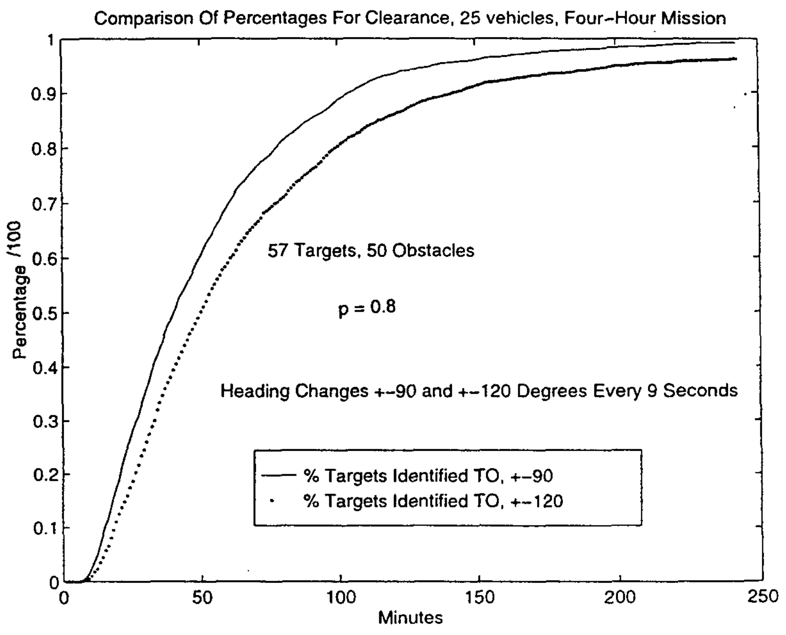


Figure 5.36 Comparison, Identification Rates, Two Ranges of Possible Heading Changes

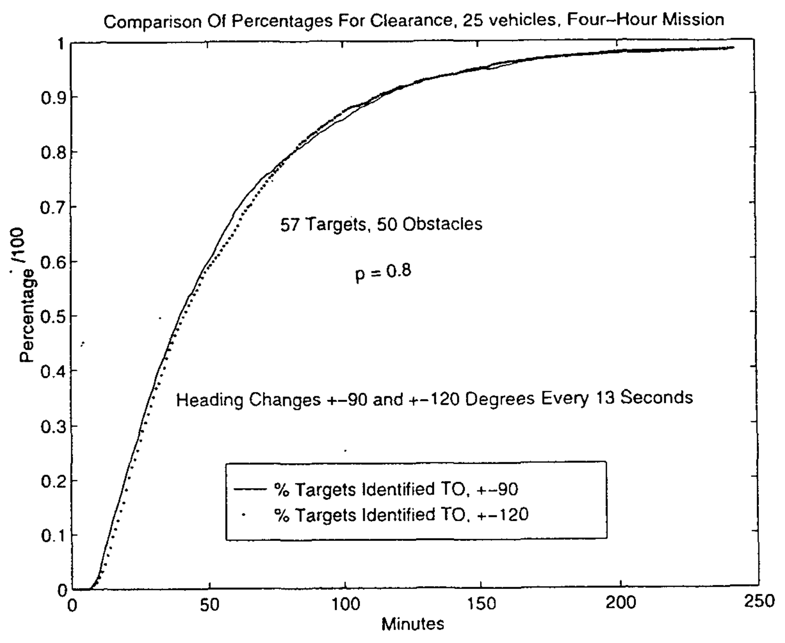


Figure 5.37 Comparison, Identification Rates, Two Ranges of Possible Heading Changes

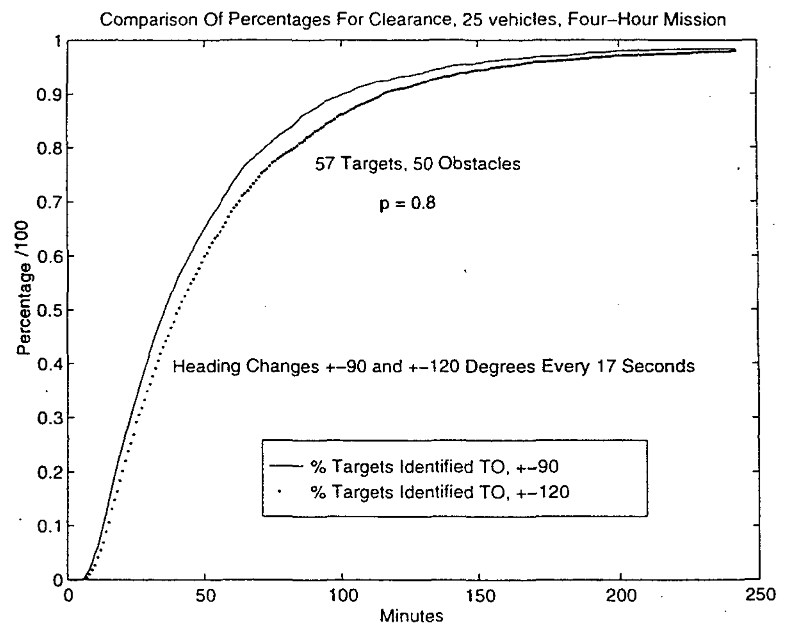
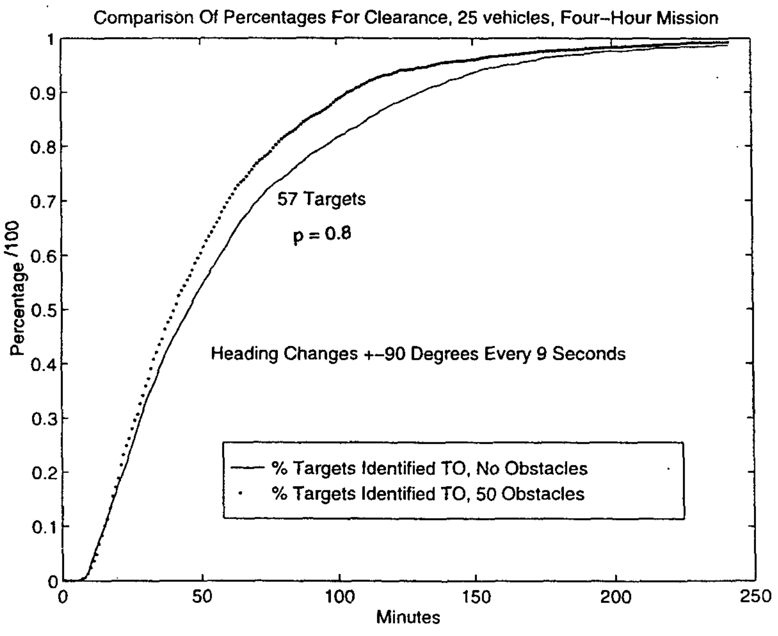
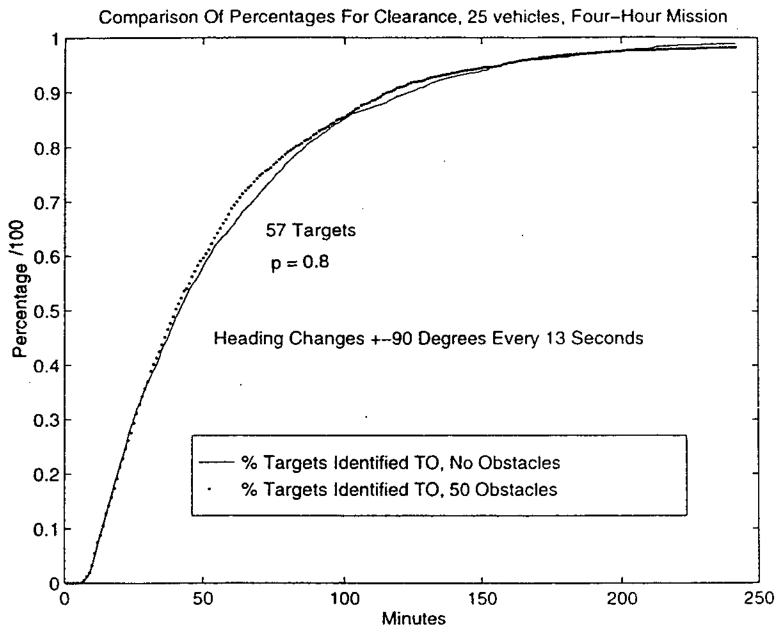


Figure 5.38 Comparison, Identification Rates, Two Ranges of Possible Heading Changes

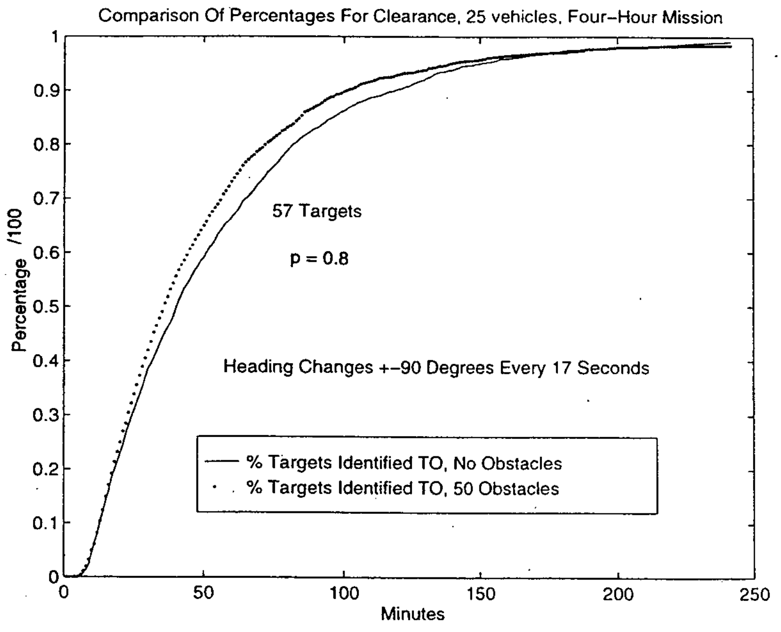
A direct comparison was also conducted between the simulations that contained the 50 obstacles and the simulations that did not contain obstacles. Specifically, the objective was to observe the impact on the identification rates of the targets by the vehicles due to the obstacles in the field, and to observe this over a range of TBHC values. The heading-change time values chosen were 9, 13 and 17 seconds. The results of the simulations indicated that by having the obstacles in the subzones, and effectively removing some of the area that normally would be searched for targets, the identification rates improved on the scenarios with obstacles over the full range of the TBHC values chosen. The results of these simulations are shown in Figures 5.39 - 5.41.



Figures 5.39 Comparison, Identification Rates, 25 Vehicles, 50 Obstacles and No Obstacles



Figures 5.40 Comparison, Identification Rates, 25 Vehicles, 50 Obstacles and No Obstacles



Figures 5.41 Comparison, Identification Rates, 25 Vehicles, 50 Obstacles and No Obstacle

The last parameter that was examined in the target and obstacle environment was the issue of the target and obstacle sensors, specifically the effect that increasing the range of these sensors would have on the identification of the targets in the approach lane. In other words, much like the TBHC value, the goal was to find the sensor range value that would give the vehicle optimum identification results, beyond which, the improvements in identification rates are only marginal. The sensors were modeled in the cookie-cutter search pattern, with the sensor ranges representing the swath radius from the center of the vehicle. The sensor ranges chosen were 2, 4 and 5 feet. Figures 5.42 and 5.43 show the results of this group of simulations. Improvements were noted on the change from a 2 ft. to a 5 ft. group of sensors, but no significant changes in the identification rates were noted on the move from a 4 ft to 5 ft. group of sensors. The results of this simulation can probably be best explained by the fact that as the sensor range increases, the ability to detect targets at a greater distance becomes greater, and with the state-based search behavior that the vehicle simulated possesses, it stops and accesses the threat much sooner than with the smaller-range sensor. If the target is evaluated as a threat, it turns 180 degrees, as mentioned in chapter three, and heads out on a random heading. The end result of all this is that, especially for areas where the targets are layered in rows, even with the sensor range extended, the vehicle will turn after it evaluates the first threat object it senses. It therefore takes, on the average, about as long to cover all the targets in the area as with a smaller sensor that, although has a shorter sensing range, is able to pass through the layered rows more often and sense and identify targets in the second and third rows. The-long range sensor, although it can see farther, is responding to the targets one at a time, and therefore is sent back away from high-density areas and has to wait for

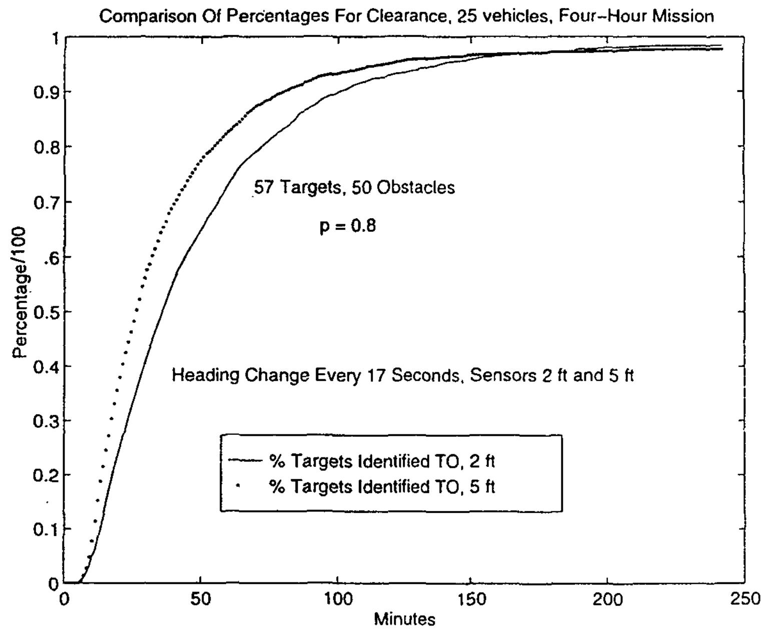


Figure 5.42 Comparison For Clearance, Sensor Ranges, Targets and Obstacles

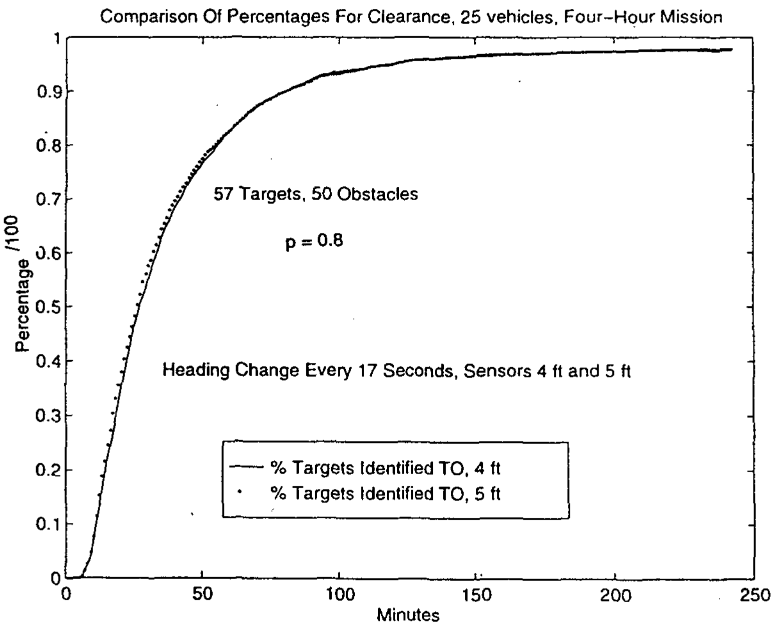


Figure 5.43 Comparison For Clearance, Sensor Ranges, Targets and Obstacles

the next arrival in the area to identify the remaining targets. In the case of the long- and short-range sensors, if the vehicle evaluates a nonthreat object, it essentially proceeds in the same direction until the next timed heading change. Figures 5.44 - 5.46 show this behavior for simulations involving sensor ranges of four and five feet. The traces clearly indicate the difficulty in the long-range sensor equipped vehicles ability to move beyond the many-layered target areas and, therefore, the reasons for the "peaking out" of the clearance rate for larger sensor ranges on this state-based search vehicle. Figure 5.47 shows the impact the increased sensor radius also has on the ability of the state-based vehicle to conduct obstacle mapping beyond the high density target areas.

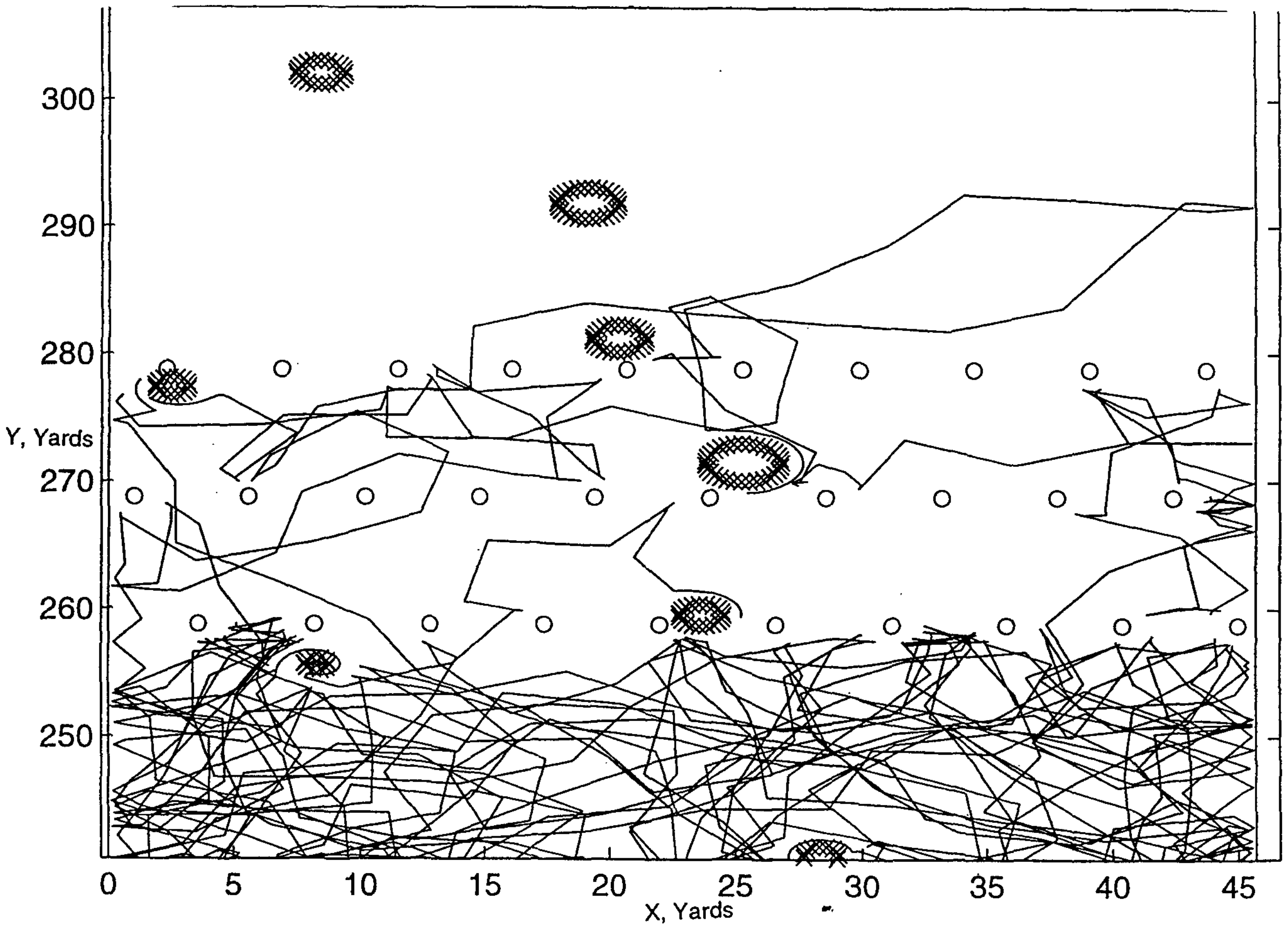


Figure 5.44 Simulation Trace In Approach Lane, Sensor Radius 5 Ft

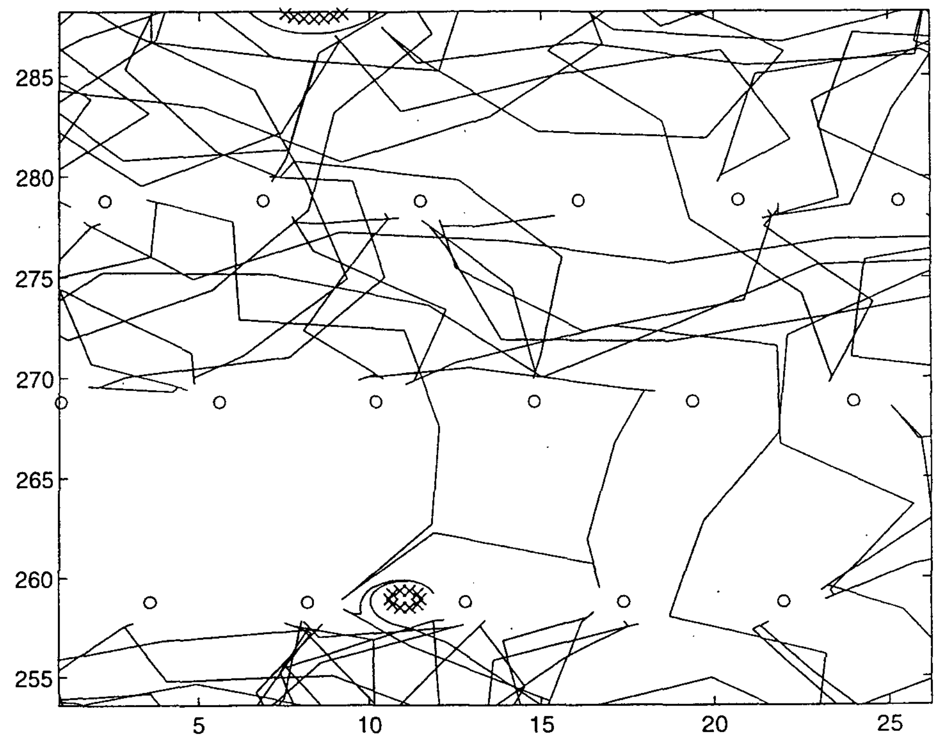


Figure 5.45 Simulation Trace In Approach Lane, Sensor Radius 4 Ft

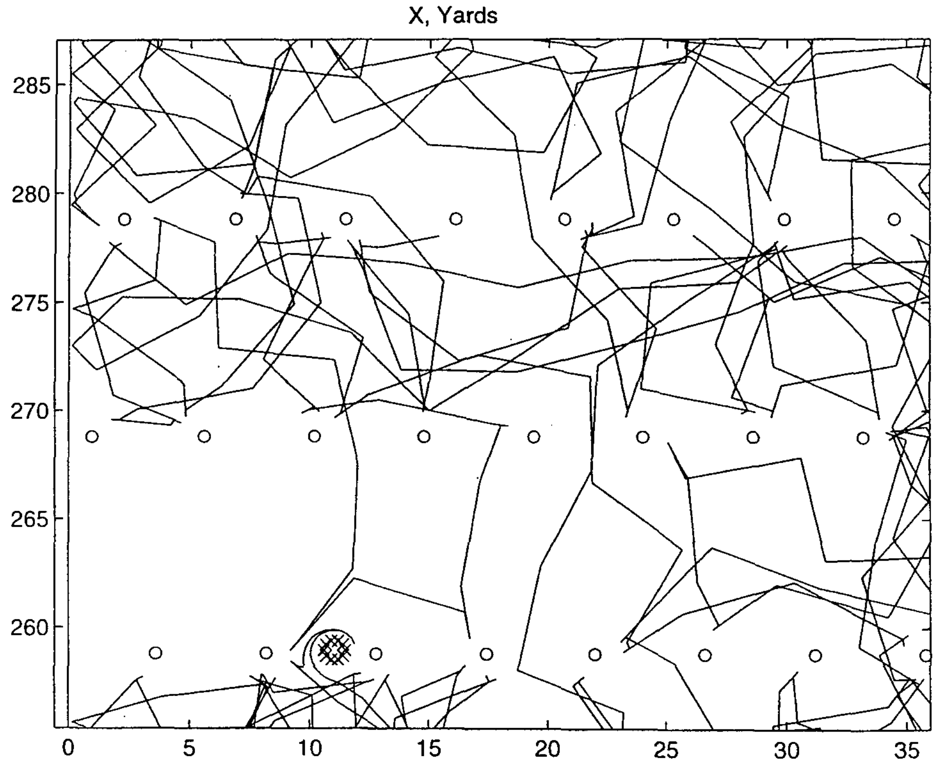


Figure 5.46 Simulation Trace In Approach Lane, Sensor Radius 4 Ft

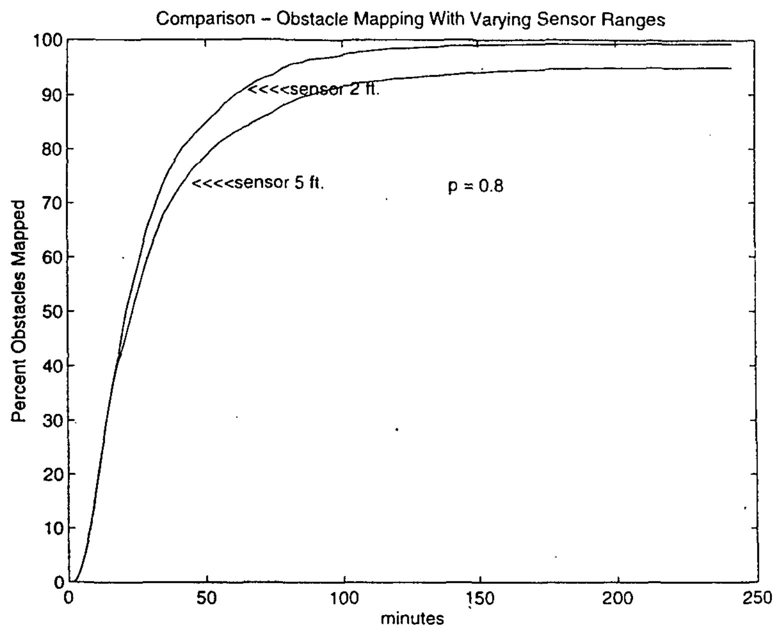


Figure 5.47 Comparison of mapping capabilties With Increased Sensor Radius, Surf Zone

VI. CONCLUSIONS

A. SUMMARY

In the course of conducting the ongoing simulations for the EOD PUCA mission and the preliminary numerical simulations for the surf-zone reconnaissance mission, some important results were observed that have given useful insight in the potential employment of autonomous vehicles in the UXO reconnaissance, identification and clearance roles. Many of the search-pattern results from the land-based scenario have proven to be useful and applicable in the surf-zone identification and reconnaissance scenario, and have allowed the simulation model to be further refined. These lessons will prove useful in future simulations as the model and development of the vehicle progresses. Specifically, these initial numerical simulations have shown that four key areas will prove to be extremely important in designing the search and obstacle avoidance routines for these autonomous vehicles: the choice of the uniformly-distributed random-heading change, the choice of the time between the random-heading changes, the choice of the appropiate sensor range based on the type of search and identification algorithm in the vehicle, and the use of combined obstacle- and boundary-avoidance routines in the vehicle, vice a single obstacle-avoidance routine, to conduct the applicable mission.

The issue of the uniformly-distributed random-heading change was simulated extensively, initially with the land-based scenario, and then carried over in to the surf-zone mission. In all simulated cases, the vehicle's target identification rates were higher for the acute-angle heading-change scenarios vice the obtuse-angle scenarios. This was especially apparent in the first hour or so of the mission profile. Specifically, the acute-angle random-heading change utilized was a heading change of +- 90 degrees from the original heading, and the obtuse-angle heading change utilized was a heading change of +- 120 degrees. It would appear the smaller range of heading changes keeps the vehicle on a more directed search, and it is also able to cover much more search area in a given mission time. The use of a larger range of heading-change angles tends to "localize" the search and reduce the identification rate, especially in the first 1 to 2 hours of the search, when performance will be the most critical. In short, for autonomous-vehicle, random-heading searches, the smaller-angle heading changes outperform the larger-angle changes in all the numerical simulations conducted to date.

Like the issue of heading changes, another parameter of the autonomous search tested extensively was the time between the random-heading changes (TBHC). The simulations conducted in the land-based scenario suggested that increasing the time between heading changes would increase the target identification rate, essentially allowing for increased area coverage. This observation was carried over to the surf-zone simulations and the TBHC value was increased to determine a favorable "no-less-than" value that would give the most desirable results for the identification and obstacle reconnaissance rate. For the surf-zone mission, any TBHC value approaching 17 seconds improved the identification rate over the previous, smaller TBHC value. After 17 seconds, however, no noticeable identification rate improvements were obtained and the 17 second value, for this approach lane geometry, seemed to the favorable "no-less-than" value for the TBHC in the search algorithm. For any autonomous-vehicle, random search, this will probably be the case, and numerical simulations need to be conducted to determine the respective "optimal" TBHC value for the given vehicle parameters and search-field geometry for any given scenario.

The most subtle observation to come out of the surf-zone simulations was the issue of an suitable sensor range based on the fact we were simulating with a vehicle with a state-based search algorithm, which "reacted" a specified way to each target detected and identified. The initial simulations conducted on the surf zone were with a 2 ft.-radius sensor, with and without obstacles, and after a sufficient data base was obtained with various configurations of the non-sensor parameters, the simulations were conducted increasing the sensor-radius range out to 5 ft. The simulations showed, much like the TBHC "no-less-than" value, that for a vehicle configured with a state-based response (detect, evaluate, turn away, drive), there will be a sensor range that, if exceeded, will not produce any significant improvement in the target identification rate. The results in chapter five showed that sensor ranges exceeding this "saturation" value tended to keep the vehicle away from the high target-density areas and very few vehicles moved beyond these areas in their respective subzones during the mission search. Although the long-range sensors compensate for this "localizing" of the search with long-range target detection, the lost capability in the mapping mission is obvious. A vehicle cannot map obstacles and conduct reconnaissance where it cannot go. Therefore, numerical simulations must be conducted to determine the best sensor range to improve the target identification rates, but not impact the ability to move around the subzones and conduct the required obstacle reconnaissance due to localizing of the search by the state-based search behavior.

The last issue that was examined was the idea of combining the obstacle-avoidance and boundary-avoidance algorithms into a single state-based "obstacle-avoidance" scheme, with the objective simplify the vehicle and produce comparable, if not better, identification results for the

UXO in the land-based scenario. Earlier simulations were conducted using state-based obstacle avoidance in the field, while using a "reflection" of the vehicle back into the field when it encountered a boundary (reflection or departure angle is the same as the angle of incidence). After completing simulations with the streamlined, all "state-based" obstacle-avoidance vehicles over a range of targets and obstacles, and comparing them with the vehicles combining boundary reflection and state-based obstacle-avoidance behavior over the same range of targets and obstacles, the "streamlined" vehicles were not found to be identify at the same rate as the original configuration. The best and most efficient obstacle and boundary avoidance tends to be obtained when they are approached as different problems to be solved. This lesson was applied to the surf zone model, and the boundary reflection and state-based obstacle mapping have worked well.

B. RECOMMENDATIONS

The principle issues for the future modeling and development of the vehicle tends to be the level of sophistication desired, and yet still be able to produce the number of vehicles needed to complete the search and reconnaissance mission. We have already shown the need for information sharing in the mapping of obstacles to complete the reconnaissance and target identification of the surf zone within a four-hour mission with 25 vehicles. This sharing of information with a central source will increase the cost and complexity, and even with this level of cooperation, many vehicles will still need to be built for the multitude of missions. This tends to show the need to continue to simulate the larger-fleet scenarios with vehicles that are fairly simple and primarily operate in a state-based format..

For the simple, state-based vehicle, which can be produced in larger numbers, the small range (1-2 ft.) target and obstacle sensor seems to be the best solution to allow the vehicles to move freely in the subzones and avoid localizing. This helps with the mapping mission, and the costs of the sensor suite should be less. Additionally, simulations should be conducted for a number of different lane geometries to develop a table of optimum sensor and TBHC values that will be utilized for a vehicle with given speed parameters. Finally, simulations should continue to examine the vehicle behavior in a number of different obstacle environments, including hedgehogs and concertina wire, to develop optimal TBHC and sensor range values for these unique environments.

APPENDIX . SURF ZONE RECONNAISSANCE SIMULATION CODE

This appendix contains the simulation code, written in the C language, that was used to run the Surf Zone Reconnaissance Mission simulations. This code can be used to simulate scenarios with and without obstacles in the surf-zone approach lane.

```
/* Bugs Testing Routine,                                        */
/* with Multple Vehicles in the PANAMA CITY test field          */
/* by a Random Search Method                                    */
/* Version 1                                                    */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define X       0
#define Y       1
#define XY      2
#define max_tgob 500

float pi, twopi,thrpi, thpi, piov2, piov4, turn_ang;

int no_veh;               /* Number of Vehicles  */
int no_target;            /* Number of Targets   */
int no_obs;               /* Number of Obstacles */
int no_rock;              /* Number of Rocks     */
int no_cleared;
int no_subarea;           /* Number of sub area  */
int no_home;              /* Number of Home      */

float max_X_length, max_Y_length;
float vh_tr_spd;     /* vehicle speed on transition            */
float vh_sch_spd;    /* vehicle speed during search            */
float ob_sensor;     /* Obstacle sencer range during transition */
float tg_sensor;     /* Target sensor distance                 */
float OO_crt;        /* Operational Obstacle criterion         */
int   hm_veh;        /* No of vehicles back to home            */
int step;            /* Time step                              */
int no_subst;        /* delt t in one sec                      */
float bb_dist;       /* BUG to BUG avid distance               */
int   dir_hch;       /* Heading change time (2 sec)            */
int   max_step;      /* Maximum allowable time step            */
int   rnd_ch_deg;    /* = 0 then -90 <= random angle <= 90     */
                     /* = 1 then -120 <= random angle <= 120   */

int tr_head_ch;      /* random search heading change interval  */
int vh_pass_tm;

struct garage {
        float pos[XY];  /* Vehicle Start Position     */
        };
struct garage *home;

struct veh_s {
        float pold[XY];  /* old position                            */
        float pnew[XY];  /* New position                            */
        float dgps[XY];  /* DGPS position                           */
        float obpo[XY];  /* Return position of Obstacle Turn        */
        float speed;     /* vehicles speed                          */
        float psi;       /* vehicles Direction                      */
        int hm_fl;       /* if hm_fl = 0 then go to assigned area   */
                         /*         = 1 then dispursing vehicles    */
                         /*         = 2 then searching target       */
                         /*         = 3 then back to Base           */
                         /*         = 4 then turn around Obstacle*/
                         /*         = 6 then stop                   */
        int trafic;      /* if traffic = 0 move                     */
```

```
                                /*              = 1 Change direction        */
                                /*              = 2 meet target & TO        */
                                /*              = 3 meet target & NTO       */
                                /*              = 4 meet Boundary           */
                                /*              = 7 meet Obstacle           */
        float obdist;           /* obstacle sensor distance                 */
        float tg_dst;           /* Target sensor distance                   */
        int wait_dch;           /* Heading change time                      */
        int wait_tm;            /* Vehicle waiting Time                     */
        int clock;              /* Vehicle start time from Home             */
        int bbo;                /* Bug to Bug  Avoid flag                   */
                                /* if = 0 continue                          */
                                /*    = 1 Stop until no BUG around it       */
        int rig;                /* No of sub region to be searched          */
        int pass;               /* vehicle passing thru time on target      */
        int to_idx;             /* Index of TO target                       */
        int nto_idx;            /* Index of target for NTO                  */
        int ob_idx;
        int ob_turn;
        int ob_stk[100];
        };
struct veh_s *vehs;

struct target {
        float pos[XY]; /* Target Position                                       */
        float gps[XY]; /* DGPS of Target                                        */
        int sch_flg;   /* If = 0 then target is not found                       */
                       /*    = 1 then target was detected at least one          */
        int no_enc;    /* Total number of encounter                             */
        int no_to;     /* Total number of TO                                    */
        int no_nto;    /* Total number of NTO                                   */
        };
struct target *tgt;

float tg_pcc;          /* Probability of correct classification of Target */
int   TC;              /* Classification procession time                  */
int   TD;              /* Communication time to report                    */
int tctd;

struct obstacle {
        float obpo[XY];
        int   found;
        };
struct obstacle *ob;

int cleartb[800];      /* It contains total # of cleared target */
int cltb_ix;           /* Index of clear table                  */
float cltb_ob[800];    /* It contains total % of found obstacles*/

struct sub_area {      /*  Containts y-coordinat of boundaries  */
        float bd;      /* of each sub area                      */
        };
struct sub_area *zone;

float Lx, Ux;

FILE *outfx, *outfy;

void  set_env(void);
float per_ob(void);
void  init_veh(void);
void  init_target(void);
void  init_ob(void);
void  move_one_sec(int);
void  chk_tg_po(int, int);
int   srh_tg(int);
int   srh_ob(int);
void  wr_xy(void);
void  move_bugs(int);
float gps_vh_err(void);
void  swap(int);
float randn(void);
void  bug_avoid(void);
float avrg(int[], int);
void  wr_av_cl(int);
void  wr_cl_dt(int *, int *, int);
void  wr_to_nto(int);
void  go_dir(int);
void  swap(int);
int   bounce_rt(int);
int   chk_base(int);
int   chk_bnd_area(int);
float rand_pi(int);
int   chk_zone(int);
void  ch_dir(int);
void  veh_dir(float, float, int );
void  wr_rst(void);
void  sort(float[], float[], int);
void  cl_pr(int, int);
int   chk_stob(int);

void set_env(void)
{
/* Define a characteristic of Vehicles */
/* and Environment of Search Field     */
/*                                     */
        int tmp;
        int j;
        float tmp1;

        vh_tr_spd = 0.6096;   /* vehicle speed on transition 2 ft/sec    */
        vh_sch_spd= 0.3048;   /* Vehicle speed during search 1 ft/sec    */
        ob_sensor = 1.2192;   /* Obstacle sencer range during transition */
        tg_sensor = 1.2192;   /* Target sensor distance                  */
        OO_crt    = 0.9144;   /* Criterion of Operational Obstacle       */
        bb_dist   = ob_sensor;
        dir_hch   = 1;        /* Vehicle turning time 2 sec              */
        vh_pass_tm= 4;        /* sensor range + 2                        */

        pi = 4.0 * atanf(1.0);
        twopi = 2.0 * pi;
        thrpi = twopi + pi;
        thpi =  thrpi / 2.0;
        piov2 = pi / 2.0;
        piov4 = piov2 / 2.0;
        turn_ang  = pi - 0.5236;


        /*  Environment Variable */
        max_X_length = 45.72;
        max_Y_length = 384.0480;
        Lx = 0.0 + vh_sch_spd;
        Ux = max_X_length - vh_sch_spd;
        no_subarea   = 5;
        if (!(zone = (struct sub_area *)malloc((no_subarea+1) * sizeof(struct s
ub_area)))) {
                fprintf(stderr, "sub_area: malloc failed\n");
        }
        tmp1 = max_Y_length / (float) no_subarea;
        zone[0].bd = 0.0;
                printf("%10.5f \n", zone[0].bd);
        for (j=1; j<no_subarea; j++) {
                zone[j].bd = zone[j-1].bd + tmp1;
                printf("%10.5f \n", zone[j].bd);
        }
        zone[no_subarea].bd = max_Y_length;
        printf("%10.5f \n", zone[no_subarea].bd);

        no_home = 2;
        if (!(home = (struct garage *)malloc(no_home * sizeof(struct garage))))
 {
                fprintf(stderr, "garage: malloc failed\n");
        }
        home[0].pos[X] = 0.0;
        home[0].pos[Y] = 0.0;
        home[1].pos[X] = max_X_length;
        home[1].pos[Y] = 0.0;

        tr_head_ch = 17;
        tg_pcc     = 0.8;
        TC         = 10;
        TD         = 10;
        tctd = TC + TD;

        max_step = 14400;
        tmp = vh_sch_spd / tg_sensor;
        no_subst = tmp + 1;
        printf(" no_subst = %d \n", no_subst);
}

main(argc, argv)
int argc;
char *argv[];
{
        char ch;
        int no_itr;
        int itr;
        int done, i;
        int stepm1, veh, no_step, tmp_ix, cltb_last;
        float cl0b_last;
        div_t dv;
        int tot_time[1000]; /* Total simulatin time for each iteration      */
        int tot_cled[1000]; /* Total no of cleared target for each iteration */

        if (argc != 3) {
            fprintf(stderr, "Usage: <program name> <no of vehcle> <no of Iterat
ion>\n");
            return 1;
        }

        no_veh = atoi(argv[1]);
        printf("No of Vehicles are %d \n", no_veh);
        no_itr = atoi(argv[2]);
        printf("No of Itrations are  %d \n", no_itr);
        if (!(vehs = (struct veh_s *)malloc(no_veh * sizeof(struct veh_s))))
                fprintf(stderr, "bounce: malloc failed\n");
                return 1;
        }

        printf("\n");
        printf("Desiered # of Rocks \n");
        scanf("%d", &no_rock);
        printf("\n");
        printf(" No of Rocks is %d \n", no_rock);

        cltb_ob[0] = 0;
        cleartb[0] = 0;

        set_env();

        printf("Desiered # of Step \n");
        printf("\n");
        printf("If Step is 0 then \n");
        printf("It will clear all target \n");
        scanf("%d", &no_step);

        printf("\n");
        printf(" no_step is %d \n", no_step);

rnd_input:
        printf("\n");
        printf("Input random change degree \n");
        printf("Type 'y' if -90 <= random angle <= 90 \n");
        printf("Type 'n' -120 <= random angle <= 120  \n");
        ch = getchar();
        printf("\n");
        if (ch == 'y' || ch == 'Y') {
                rnd_ch_deg = 0;
                printf("Vehicle will turn -90 <= random angle <= 90  \n");
        }
        else if (ch == 'n' || ch == 'N') {
                rnd_ch_deg = 1;
                printf("Vehicle will turn -120 <= random angle <= 120  \n");
        }
        else {
                printf("Error. Type again  \n");
                goto rnd_input;
        }

        /* Initilize the targets   */
        init_target();

        for (itr=1; itr<=no_itr; itr++)
        {
            /* Initilize the obstacles */
            init_ob();

            printf("$$$$$$$$$ %d %d \n", itr, no_obs);
            /* Initilize  search flag for targets   */
            for (i=0; i<no_target; i++)
                tgt[i].sch_flg = 0;
            for (i=0; i<no_obs; i++)
                ob[i].found = 0;

            init_veh();
```

```
/* wr_rst(); */

if (itr == no_itr) {
    outfx = fopen("x.out","w");
    outfy = fopen("y.out","w");
    wr_xy();
}

no_cleared = 0;
hm_veh = 0;
done = 0;
step = 1;

while(!done) {

    for (veh=0; veh<no_veh; veh++) {

        vehs[veh].clock++;

        if (vehs[veh].clock <= 0)
            continue;

        move_one_sec(veh);

        if (vehs[veh].hm_fl == 2 ) {
            if (((vehs[veh].clock % tr_head_ch) == 0)
                    && (vehs[veh].trafic == 0)) {
                vehs[veh].trafic = 1;
                vehs[veh].wait_dch = 0;
            }

            /* Random Searching a given target */
        }
    }

    bug_avoid();

    if (itr == no_itr)
            wr_xy();

    if (no_step == 0 ){
        if (no_cleared == no_target || step > max_step)
            done = 1;
    }
    else {
        if ((step >= no_step ) || (hm_veh == no_veh))
            done = 1;
    }

    dv = div(step, 60);
    if (dv.rem == 0) {
        if (itr == 1) {
            cleartb[dv.quot] = no_cleared;
            cltb_ob[dv.quot] = per_ob();
        }
        else {
            if (dv.quot <= cltb_ix) {
                cleartb[dv.quot] += no_cleared;
                cltb_ob[dv.quot] += per_ob();
            }
            else {
                cleartb[dv.quot] = cltb_last + no_cleared;
```

```
                        cltb_ob[dv.quot] = cl0b_last + per_ob();
                    }
                }
            }


            step++;
        }
        if (itr == no_itr) {
            fclose(outfx);
            fclose(outfy);
        }

        dv = div(step-1, 60);
        tmp_ix = dv.quot+1;
        if (dv.rem == 0) {
            if (itr == 1)
                cltb_ix = dv.quot;
            else {
                if (dv.quot < cltb_ix)
                    for (i=tmp_ix; i<= cltb_ix; i++) {
                        cleartb[i] += no_cleared;
                        cltb_ob[i] += per_ob();
                    }
                else
                    cltb_ix = dv.quot;
            }
        }
        else {
            if (itr == 1) {
                cleartb[tmp_ix] = no_cleared;
                cltb_ob[tmp_ix] = per_ob();
                cltb_ix = tmp_ix;
            }
            else {
                if (tmp_ix <= cltb_ix) {
                    for (i=tmp_ix; i<= cltb_ix; i++) {
                        cleartb[i] += no_cleared;
                        cltb_ob[i] += per_ob();
                    }
                }
                else {
                    cleartb[tmp_ix]  = cleartb[dv.quot];
                    cltb_ob[tmp_ix]  = cltb_ob[dv.quot];
                    cltb_ix = tmp_ix;
                }
            }
        }
        cltb_last = cleartb[cltb_ix];
        cl0b_last = cltb_ob[cltb_ix];

        tot_time[itr-1] = step;
        tot_cled[itr-1] = no_cleared;

        printf("%5d %7d \n", no_cleared, step);
    }

    printf("average time %5.f \n", avrg(tot_time, no_itr));
    printf("average cleared targets %10.3f \n",  avrg(tot_cled, no_itr));

    wr_av_cl(no_itr);
    wr_cl_dt(tot_time, tot_cled, no_itr);
```

```
        wr_to_nto(no_itr);
}

float per_ob()
{
/*                                            */
/*   Calculate the clearance rate of obstacle */
/*                                            */
        int i, tot;

        tot = 0;
        for (i=0; i<no_obs; i++) {
            if (ob[i].found)
                tot++;
        }

        return(((float) tot)/ ((float) no_obs) * 100.0);
}
void wr_rst()
{
        int i;
        for ( i=0; i<no_veh; i++)
        {
                printf("vehs = %d \n", i);
                printf("vehs[i].pold[X] = %f \n", vehs[i].pold[X]);
                printf("vehs[i].pold[Y] = %f \n", vehs[i].pold[Y]);
                printf("vehs[i].pnew[X] = %f \n", vehs[i].pnew[X]);
                printf("vehs[i].pnew[Y] = %f \n", vehs[i].pnew[Y]);
                printf("vehs[i].dgps[X] = %f \n", vehs[i].dgps[X]);
                printf("vehs[i].dgps[Y] = %f \n", vehs[i].dgps[Y]);
                printf("vehs[i].speed   = %f \n", vehs[i].speed);
                printf("vehs[i].hm_fl   = %d \n", vehs[i].hm_fl);
                printf("vehs[i].trafic  = %d \n", vehs[i].trafic);
                printf("vehs[i].bbo     = %d \n", vehs[i].bbo);
                printf("vehs[i].clock   = %d \n", vehs[i].clock);
                printf("vehs[i].rig     = %d \n", vehs[i].rig);
        }
}


void init_veh()
/*------------------------------------------------------------------------ */
/*    Function:  init_veh                                                  */
/*    Parameters:                                                          */
/*    set the initial position for each vehcles                            */
/*    Get the target for searching and define searching area               */
/*------------------------------------------------------------------------ */
{
        float dy, dx, tmp, tmp_x, tmp_y;
        int i, idx;
        int qut;
        int no_subarea_veh;

        no_subarea_veh = no_veh / no_subarea;
        for ( i=0; i<no_veh; i++)
        {
                qut = i / no_subarea_veh + 1;
                vehs[i].rig     = no_subarea - qut;
                if ((i % 2) == 0) {
                    tmp_x = home[0].pos[X];
                    tmp_y = home[0].pos[Y];
                }
                else {
                    tmp_x = home[1].pos[X];
                    tmp_y = home[1].pos[Y];
                }
                vehs[i].pold[X] = vehs[i].pnew[X] = tmp_x;
                vehs[i].pold[Y] = vehs[i].pnew[Y] = tmp_y;
                vehs[i].bbo     = 0;
                vehs[i].trafic  = 0;
                vehs[i].obdist  = ob_sensor;
                vehs[i].tg_dst  = tg_sensor;
                vehs[i].psi     = piov2;
                vehs[i].speed   = vh_tr_spd / no_subst;
                vehs[i].hm_fl   = 0;
                qut             = i / 2;
                vehs[i].clock   = qut * (-3);
                vehs[i].ob_idx  = 0;
        }
}

void move_one_sec(int v)
{
        int stp;
        int don, sub_stp;
        int flag, index, idx;

        don  = 0;
        stp = no_subst;

        if (vehs[v].bbo || (vehs[v].hm_fl == 6)) {
                vehs[v].bbo = 0;
                don = 1;
        }
        else {
            switch ( vehs[v].trafic){
                case 0:
                    sub_stp = 1;
                    break;
                case 1:
                    vehs[v].wait_dch++;
                    if (vehs[v].wait_dch <= dir_hch)
                        don = 1;
                    else {
                        ch_dir(v);
                        sub_stp = 1;
                        vehs[v].trafic = 0;
                    }
                    break;
                case 2:
                    /* One step back and turn right */
                    vehs[v].wait_dch++;
                    if (vehs[v].wait_dch < vehs[v].wait_tm)
                        don = 1;
                    else {
                        don = 1;
                        swap(v);
                        vehs[v].trafic = 1;
                        vehs[v].wait_dch = 0;
                        vehs[v].psi += pi;
                    }
                    break;
                case 3:
                    vehs[v].wait_dch++;
        if (vehs[v].wait_dch < vehs[v].wait_tm)
            don = 1;
        else {
            sub_stp = 1;
            vehs[v].trafic = 5;
            vehs[v].pass   = 0;
        }
        break;
    case 4:
        vehs[v].wait_dch++;
        if (vehs[v].wait_dch <= dir_hch)
            don = 1;
        else {
            sub_stp = 1;
            vehs[v].trafic = 0;
        }
        break;
    case 5:
        vehs[v].pass++;
        sub_stp = 1;
        if (vehs[v].pass > vh_pass_tm)
            vehs[v].trafic = 0;
        break;
    case 6:
        vehs[v].wait_dch++;
        if (vehs[v].wait_dch <= dir_hch)
            don = 1;
        else {
            vehs[v].ob_turn++;
            sub_stp = 1;
            vehs[v].wait_dch = 0;
            vehs[v].trafic = 7;
        }
        break;
    case 7:
        vehs[v].wait_dch++;
        if (vehs[v].wait_dch < dir_hch)
            don = 1;
        else {
            vehs[v].psi -= piov2;
            vehs[v].wait_dch = 0;
            sub_stp = 1;
        }
        break;
    case 8:
        /* One step back and turn right */
        vehs[v].wait_dch++;
        if (vehs[v].wait_dch <= vehs[v].wait_tm)
            don = 1;
        else {
            sub_stp = 1;
            vehs[v].trafic = 6;
            vehs[v].wait_dch = 0;
        }
        break;
    case 9:
        /* One step back and turn right */
        vehs[v].wait_dch++;
        if (vehs[v].wait_dch < vehs[v].wait_tm)
            don = 1;
        else {
            sub_stp = 1;
```

```
                    vehs[v].trafic = 1;
                    vehs[v].wait_dch = 0;
                    vehs[v].psi += piov2;
                }
                break;
            default:
                sub_stp = 1;
                break;
        }
    }
    if (vehs[v].psi >= twopi)
            vehs[v].psi -= twopi;
    else if (vehs[v].psi < 0.0)
            vehs[v].psi += twopi;

    while (!don)
    {
        if (vehs[v].hm_fl != 0) {
            /* Check Obstacle */
            idx = srh_ob(v);
            if (idx >= 0) {
                don = 1;
                vehs[v].wait_dch = 0;

                if (vehs[v].hm_fl == 4) {
                    vehs[v].psi +=  0.8727;
                    if (vehs[v].psi >= twopi)
                        vehs[v].psi -= twopi;
                    while (srh_ob(v) >= 0) {
                        vehs[v].psi +=  0.8727;
                        if (vehs[v].psi >= twopi)
                            vehs[v].psi -= twopi;
                    }
                    vehs[v].trafic = 6;
                }
                else {
                    vehs[v].wait_tm = dir_hch;
                    vehs[v].trafic = 2;
                }
            }

           /*  check a target */
            index = srh_tg(v);
            if (index >= 0)
            {
                don = 1;
                if (vehs[v].hm_fl == 4) {
                    vehs[v].wait_dch = 0;
                    if (index == vehs[v].to_idx) {
                         chk_tg_po(index, v);
                         vehs[v].trafic = 6;
                    }
                    else if (index == vehs[v].nto_idx)
                         vehs[v].trafic = 6;
                    else {
                         cl_pr(index, v);
                         chk_tg_po(index, v);
                         vehs[v].trafic = 8;
                    }
                }
                else {
                    vehs[v].wait_dch = 0;
```

```
                        don = 1;
                        if (vehs[v].hm_fl == 1)
                            vehs[v].hm_fl = 2;

                        cl_pr(index, v);
                    }
                }

        }

        if (!don) move_bugs(v);

        switch (vehs[v].hm_fl) {
            case 0:
                /* zone Area Check */
                if (chk_zone(v)) {
                    vehs[v].trafic = 0;
                    don = 1;
                }
                break;
            case 1:
                /* Dispuring the vehicle in the zone */
                if (vehs[v].clock >= 50) {
                    don = 1;
                    vehs[v].hm_fl = 2;
                    vehs[v].trafic = 1;
                    vehs[v].wait_dch = 0;
                }
                break;
            case 2:
                /* serching targrts    */
                if (chk_bnd_area(v) ) {
                    vehs[v].wait_dch = 0;

                    if (vehs[v].trafic == 2)
                            break;

                    if (vehs[v].trafic == 3)
                            vehs[v].wait_tm = TC + 1;
                    else
                            vehs[v].trafic = 4;
                    don = 1;
                }
                break;
            case 3:
                /* Check home base   */
                if (chk_base(v))
                    don = 1;
                    break;
            case 4:
                /* Check starting pts 0f obstacle */
                if (vehs[v].ob_turn > 5) {
                    flag = chk_stob(v);
                    if (flag != 0) {
                        vehs[v].wait_tm = TC;
                        vehs[v].trafic = 9;
                    }
                }
            default:
                don = 1;
                break;
        }
```

```
                sub_stp++;
                if (sub_stp > stp)
                    don = 1;
            }

}

void chk_tg_po(int index, int v)
{
        float xmx, ymy;
        float theta, tmp, tmp1;
        float the1, tmp2;

        the1 = vehs[v].psi;
        if (the1 < 0.0)
                the1 += twopi;

        xmx = tgt[index].pos[X] - vehs[v].pnew[X];
        ymy = tgt[index].pos[Y] - vehs[v].pnew[Y];
        theta = atan2f(ymy, xmx);
        if (theta < 0.0)
            theta += twopi;
        tmp = fabsf(theta - the1);
        tmp2 = fabsf(tmp - twopi);
        if ((tmp <= piov4 || tmp2 <= piov4)) {
            vehs[v].psi = theta + 0.8727;
            if (vehs[v].psi >= twopi)
                vehs[v].psi -= twopi;
        }

}

int chk_stob(int v)
/*                                                      */
/* Check if the Vehicle return to starting position     */
/*                                                      */
{
        int flg, k, idx;
        float x1, y1, x2, y2;
        float lx, ux, ly, uy;
        float diam, re_diam, x3, y3, x4, y4;
        float min_x, max_y, min_y, max_x;

        flg = 0;

        x2 = vehs[v].pnew[X];
        y2 = vehs[v].pnew[Y];
        x1 = vehs[v].obpo[X];
        y1 = vehs[v].obpo[Y];
        lx = x1 - ob_sensor;
        ux = x1 + ob_sensor;
        ly = y1 - ob_sensor;
        uy = y1 + ob_sensor;
        x3 = x2;
        y3 = y2;

        if ((x2 > lx) && (x2 < ux) && (y2 > ly) && (y2 < uy)) {
            idx = vehs[v].ob_stk[0];
            x2  = ob[idx].obpo[X];
            y2  = ob[idx].obpo[Y];
            min_x = max_x = x2;
```

```
min_y = max_y = y2;
for (k=1; k<vehs[v].ob_idx; k++) {
    idx = vehs[v].ob_stk[k];
    x1 = ob[idx].obpo[X];
    if (x1 < min_x)
        min_x = x1;
    if (x1 > max_x)
        max_x = x1;

    y1  = ob[idx].obpo[Y];
    if (y1 < min_y)
        min_y = y1;
    if (y1 > max_y)
        max_y = y1;
}

x1 = fabsf(max_x - min_x);
y1 = fabsf(max_y - min_y);

if (x1 > y1)
    diam = x1;
else
    diam = y1;

/* printf("diam = %10.3f \n", diam);*/
x1 =  min_x - 0.15;
x2 =  max_x + 0.15;
y1 =  min_y - 0.15;
y2 =  max_y + 0.15;
min_x = 99999.0;
max_x = 0.0;
min_y = 99999.0;
max_y = 0.0;

for (k=0; k < no_obs; k++) {
    x4 = ob[k].obpo[X];
    if ((x4 > x1) && (x4 < x2)) {
        y4 = ob[k].obpo[Y];
        if ((y4 > y1) && (y4 < y2)) {
            ob[k].found = 1;
            if (x4 < min_x)
                min_x = x4;
            if (x4 > max_x)
                max_x = x4;
            if (y4 < min_y)
                min_y = y4;
            if (y4 > max_y)
                max_y = y4;
        }
    }
}

x1 = fabsf(max_x - min_x);
y1 = fabsf(max_y - min_y);

if (x1 > y1)
    re_diam = x1;
else
    re_diam = y1;

/* printf(" v = %5d, found Obstacle at step = %5d", v, step ); */
/*printf(" position ( %7.2f,%7.2f) with diameter %7.3f \n", x3, y3,
re_diam); */

            vehs[v].ob_idx = 0;
            vehs[v].hm_fl  = 2;

            if (diam >= OO_crt) {
                flg = 1;
                /* printf("     Classified as OO  with diameter %7.3f \n", di
);*/
            }
            else {
                flg = 2;
                /* printf("     Classified as NOO with diameter %7.3f \n", di
); */
            }
        }

        return(flg);
}

void move_bugs(int v)
{
        int ip1, im1;
        float dx, dy;
        int sign;

        vehs[v].pold[X] = vehs[v].pnew[X];
        vehs[v].pold[Y] = vehs[v].pnew[Y];
        if (vehs[v].hm_fl != 6)
        {
                dx = vehs[v].speed * cos(vehs[v].psi);
                dy = vehs[v].speed * sin(vehs[v].psi);
                vehs[v].pnew[X] += dx;
                vehs[v].pnew[Y] += dy;

                vehs[v].dgps[X] = vehs[v].pnew[X] + gps_vh_err();
                vehs[v].dgps[Y] = vehs[v].pnew[Y] + gps_vh_err();
        }
}

float gps_vh_err()
/*------------------------------------------------------------------- */
/*    Function:  gps_vh_err*/
/*    Parameters: */
/*    Returns: value |vehicle DGPS error| <= 0.14 */
/*------------------------------------------------------------------- */
{
        float err;
        do {
                err = randn();
        } while ( fabsf(err) > 1.0 );
       return(err*0.14);
}

void cl_pr(int index, int v)
/*------------------------------------------------------------------- */
/*    Function:  cl_pr                                                 */
/*    Parameters:                                                      */
/*    Returns: value [0, 1]                                            */
/*------------------------------------------------------------------- */
{
        float err;
        err = (float) drand48();
        tgt[index].no_enc ++;
        if (err <= tg_pcc) {
                if (tgt[index].sch_flg == 0) {
                        no_cleared ++;
                        tgt[index].sch_flg = 1;
                }
                /*
                printf(" v = %5d, step = %5d, tg = %5d, TO \n", v, step, index)
;
                */
                vehs[v].trafic = 2;

                vehs[v].wait_tm = tctd;
                tgt[index].no_to ++;
                vehs[v].to_idx = index;
        }
        else {
                if ( !(vehs[v].trafic == 2))
                    vehs[v].trafic = 3;

                vehs[v].wait_tm = TC;
                tgt[index].no_nto++;
                vehs[v].nto_idx = index;
                /*
                printf(" v = %5d, step = %5d, tg = %5d, NTO \n", v, step, index
);
                */
        }
}


float randn()
/*----------------------------------------------------------------------- */
/*    Function: randn */
/*    Summary: taken from gasdev() in Numerical Recipes in C */
/*    Parameters: */
/*    Returns: guass dstributed random value */
/*----------------------------------------------------------------------- */
{
        static int iset = 0;
        static float gset;

        float fac,r,v1,v2;

    if (!iset)
    {
        do
        {
                v1 = 2.0*(float) drand48() - 1.0;
                v2 = 2.0*(float) drand48() - 1.0;
                r = v1*v1 + v2*v2;
        } while ((r >= 1.0) || (r == 0.0));

        fac = sqrtf(-2.0*log(r)/r);

        iset = 1;
        gset = v1*fac;
        return (v2*fac);
    }
    else
    {
        iset = 0;
        return (gset);
    }
}

int srh_tg(int i)
{
        int idx, k;
        float xmx, ymy, dist;
        float x2, y2;
        float min_dist, dt;
        float lx, ly, ux, uy, tx, ty;

        idx = -1;
        dist = tg_sensor + 0.000005;
        min_dist = dist;

        x2 = vehs[i].pnew[X];
        y2 = vehs[i].pnew[Y];
        lx = x2 - dist;
        ly = y2 - dist;
        ux = x2 + dist;
        uy = y2 + dist;

        for (k=0; k<no_target; k++)
        {
                if ((vehs[i].trafic == 6) && (k == vehs[i].nto_idx))
                    continue;

                tx = tgt[k].pos[X];
                ty = tgt[k].pos[Y];
                if ((tx > lx) && (tx < ux) && (ty > ly) && (ty < uy))
                {
                        xmx = x2 - tx;
                        ymy = y2 - ty;
                        dt = sqrtf(xmx * xmx + ymy * ymy);
                        if (dt < min_dist)
                        {
                            min_dist = dt;
                            idx = k;
                        }
                }
        }
        return (idx);
}

int srh_ob(int i)
{
        int ix, k;
        float min, dist, xmx, ymy;
        float theta, piover4, tmp, min_dt, tmp1;
        float the1, tmp2;

        ix = -1;
        min = vehs[i].obdist;
        min_dt = min + 0.00005;
        the1 = vehs[i].psi;
        if (the1 < 0.0)
                the1 += twopi;
        for (k=0; k<no_obs; k++)
        {
            xmx = ob[k].obpo[X] - vehs[i].pnew[X];
            ymy = ob[k].obpo[Y] - vehs[i].pnew[Y];
            if (fabsf(xmx) <= min && fabsf(ymy) <= min)
            {
                theta = atan2f(ymy, xmx);
                if (theta < 0.0)
                        theta += twopi;
                dist = sqrtf(xmx * xmx + ymy * ymy);
                tmp = fabsf(theta - the1);
                tmp2 = fabsf(tmp - twopi);
                if ((tmp <= piov4 || tmp2 <= piov4) && (dist <= min))
                    if ( dist < min_dt)
                    {
                        min_dt = dist;
                        ix = k;
                        tmp1 = theta;
                    }
            }
        }
        if (ix >= 0) {
            vehs[i].psi = tmp1;
            if (!ob[ix].found) {
                ob[ix].found = 1;
                k = vehs[i].ob_idx;
                vehs[i].ob_stk[k] = ix;
                if (k == 0) {
                    vehs[i].hm_fl  = 4;
                    vehs[i].ob_turn = 0;
                    vehs[i].obpo[X] = vehs[i].pnew[X];
                    vehs[i].obpo[Y] = vehs[i].pnew[Y];
                }
                vehs[i].ob_idx++;
            }
        }

        return (ix);
}

void wr_xy(void)
{
        int v;

        for (v=0; v<no_veh; v++)
                fprintf(outfx, "%10.5f", vehs[v].pnew[X] );
        fprintf(outfx, "\n");

        for (v=0; v<no_veh; v++)
                fprintf(outfy, "%10.5f", vehs[v].pnew[Y]);
        fprintf(outfy, "\n");
}

void bug_avoid(void)
{
/*      BUG to BUG Avoidance Alogorithm    */
/*      based on "rank" to prevent the     */
/*      collision                          */
/*                                         */
        int j, k;
        float dx, dy;
```

```
          for (j=0; j<no_veh-1; j++)
          {
              if (vehs[j].hm_fl != 6)
                  for (k=j+1; k<no_veh; k++) {
                      if (((vehs[k].hm_fl != 6) || !( vehs[k].bbo)) && (vehs[k
lock >= 0)) {
                          dy = vehs[j].pnew[Y] - vehs[k].pnew[Y];
                          dx = vehs[j].pnew[X] - vehs[k].pnew[X];
                          if ((fabsf(dy) < bb_dist) && (fabsf(dx) < bb_dist))
                              if (sqrtf(dx*dx + dy*dy) <= bb_dist)
                                  vehs[k].bbo = 1;
                      }
                  }
          }
}

float avrg(int tm[], int size)
{
          /* Compute mean value of a list */

          int j, sum;

          sum = 0;
          for (j=1; j<size; j++)
                  sum = sum + tm[j] - tm[0];

          return((float) sum / ((float) size) + (float) tm[0]);
}
void wr_av_cl(int size)
{
          div_t dv;
          char ch[13];
          int j, x, y;
          float tmp, tmp1;

          FILE *inptr;

          strcpy(ch, "pan30000.out");
          x = no_veh;

          if (x < 10 )
                  ch[7] = ch[7] + x;
          else if (x < 100) {
                  dv = div(x,10);
                  ch[6] = ch[6] + dv.quot;
                  ch[7] = ch[7] + dv.rem;
          }
          else {
                  dv = div(x,100);
                  ch[5] = ch[5] + dv.quot;
                  y = dv.rem;
                  dv = div(y,10);
                  ch[6] = ch[6] + dv.quot;
                  ch[7] = ch[7] + dv.rem;
          }


          inptr = fopen(ch,"w");
          for (j=0;j<=cltb_ix ;j++) {
                  tmp = ((float) cleartb[j]) / ((float) size);
                  tmp1 = cltb_ob[j] / ((float) size);
                  fprintf(inptr, "%10.2f  %10.2f\n", tmp, tmp1 );
```

```
        }
        fclose(inptr);
}

void wr_cl_dt(int *ttm, int *tcl, int ittr)
{
        div_t dv;
        char ch[13];
        int j, x, y;
        float tmp;

        FILE *inptr;

        strcpy(ch, "pancr000.out");
        x = no_veh;

        if (x < 10 )
                ch[7] = ch[7] + x;
        else if (x < 100) {
                dv = div(x,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }
        else {
                dv = div(x,100);
                ch[5] = ch[5] + dv.quot;
                y = dv.rem;
                dv = div(y,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }


        inptr = fopen(ch,"w");
        for (j=0;j<ittr ;j++) {
                fprintf(inptr, "%5d %5d %7d \n", j+1, tcl[j], ttm[j]);
        }
        fclose(inptr);
}


void go_dir(int v)
{
        int idx;
        switch (vehs[v].hm_fl) {
            case 0:
                break;
            case 1:
                break;
            case 2:
                break;
            case 3:
                if ((v % 2) == 0)
                        veh_dir(home[0].pos[X], home[0].pos[Y], v);
                else
                        veh_dir(home[1].pos[X], home[1].pos[Y], v);
                break;
            default:
                break;
        }
}
```

```
void veh_dir(float gpsx, float gpsy, int v)
{
/*                                                                    */
/*      Compute the vehicle direction from  vehicle position to target */
/*                                                                    */
        int k;
        float ang;
        ang = atan2f(gpsy - vehs[v].dgps[Y], gpsx - vehs[v].dgps[X]);
        if (ang < 0.0)
                vehs[v].psi = ang + twopi;
        else
                vehs[v].psi = ang;
}


void ch_dir(int v)
{
/*      Change the v vehicle direction          */
/*      Turnning angle is uniform random between */
/*      -120 and +120 degree                    */

        float tp;

        if (rnd_ch_deg)
                tp = vehs[v].psi + rand_pi(2);
        else
                tp = vehs[v].psi + rand_pi(1);

        if (tp < 0.0)
                vehs[v].psi = twopi + tp;
        else if (tp >= twopi)
                vehs[v].psi = tp - twopi;
        else
                vehs[v].psi = tp;


}


void swap(int v )
{
        float tmpx, tmpy;

        tmpx = vehs[v].pnew[X];
        tmpy = vehs[v].pnew[Y];
        vehs[v].pnew[X] = vehs[v].pold[X];
        vehs[v].pnew[Y] = vehs[v].pold[Y];
        vehs[v].pold[X] = tmpx;
        vehs[v].pold[Y] = tmpy;
}

float rand_pi(int kk )
{
/*                                                       */
/* return non-negative floating-point values .           */
/* uniformly distributed over the interval [-120, 120]. */
/*                                                       */
        float tmp;

        switch (kk) {
            case 1:
                return (pi * ((float) drand48() - 0.5));
                break;
            case 2:
                return ( pi * ((float) drand48() - 0.5) * 4.0 / 3.0);
                break;
            case 3:
                tmp = pi * (float) drand48() * 0.5;
                return(0.7778 * tmp + 0.1745);
                break;
            default:
                break;
        }
}

int chk_zone(int v)
{
        /* Check zone area */
        float xp, yp, ang;
        int flg, zn;

        xp = vehs[v].pnew[X];
        yp = vehs[v].pnew[Y];
        flg = 0;

        zn = vehs[v].rig;

        if (yp >= zone[zn].bd) {
                flg = 1;
                ang = rand_pi(3);
                vehs[v].hm_fl  = 1;
                vehs[v].speed   = vh_sch_spd / no_subst;
                vehs[v].clock = 1;
                if ((v % 2) == 0)
                    vehs[v].psi = ang;
                else
                    vehs[v].psi += ang;
        }

        return(flg);
}


int chk_bnd_area(int v)
{
        float xp, yp;
        int flg, jd;
        float tmp;
        float Ly, Uy;
        float  lx05, ux05, ly05, uy05, dirt;

        /* Turn from Upper boundary */
        xp = vehs[v].pnew[X];
        yp = vehs[v].pnew[Y];
        jd = vehs[v].rig;
        Ly = zone[jd].bd + vh_sch_spd;
        Uy = zone[jd+1].bd - vh_sch_spd;
        dirt = vehs[v].psi;
        flg = 1;
        if (dirt < 0.0)
                dirt += twopi;
        else if(dirt >= twopi)
                dirt -= twopi;
        if  (xp <= Lx ) {
                if ((dirt >= piov2) && (dirt <= thpi))
                    tmp =  thrpi - dirt;
                else
                    tmp = dirt;
        }
        else if (xp >= Ux) {
                if (((dirt >= 0.0) && (dirt <= piov2)) || ((dirt >= thpi) &&
irt <= twopi)))
                    tmp =  thrpi - dirt;
                else
                    tmp = dirt;
                }
        else if (yp <= Ly) {
                if ((dirt >= pi) && (dirt <= twopi))
                    tmp = twopi - dirt;
                else
                    tmp = dirt;
                }
        else if (yp >= Uy) {
                if ((dirt >= 0.0) && (dirt <= pi))
                    tmp = twopi - dirt;
                else
                    tmp = dirt;
                }
        else
                flg = 0;

        if (flg) {
            if (tmp < 0.0)
                vehs[v].psi = twopi + tmp;
            else if (tmp >= twopi)
                vehs[v].psi = tmp - twopi;
            else
                vehs[v].psi = tmp;
        }
        return(flg);
}

int chk_base(int v)
{
        int flg;
        float y1, x1;

        flg = 0;
        if ((v % 2) == 0) {
                x1 = fabsf(vehs[v].dgps[X] - home[0].pos[X]);
                y1 = fabsf(vehs[v].dgps[Y] - home[0].pos[Y]);
        }
        else {
                x1 = fabsf(vehs[v].dgps[X] - home[1].pos[X]);
                y1 = fabsf(vehs[v].dgps[Y] - home[1].pos[Y]);
        }
        if (x1 <= 1.0 && y1 <= 1.0)
        {
                flg = 1;
                vehs[v].hm_fl = 6;
                hm_veh ++;
        }
        return(flg);
```

```
}

void init_target()
/*------------------------------------------------------------------- */
/*    Function:  init_target                                           */
/*    Parameters:                                                      */
/*    set the initial target position                                  */
/*------------------------------------------------------------------- */
{
        int i;
        float x1, y1, tgt_x[500], tgt_y[500];
        float dx, dy, tmp, dist;
        FILE *inptr;
/*                                  */
/*  Initialize the targets position */
/*                                  */

        inptr = fopen("target.dat","r");
        i = 0;
        while (fscanf(inptr, "%f %f", &x1, &y1) !=EOF)
        {
                tgt_x[i] = x1;
                tgt_y[i] = y1;
                i++;
        }
        no_target = i;
        fclose(inptr);
        if (!(tgt = (struct target *)malloc(no_target * sizeof(struct target)))
) {
                fprintf(stderr, "bounce: malloc failed in init_target\n");
        }

        sort(tgt_x, tgt_y, no_target);

        for (i=0; i<no_target; i++)
        {
                tgt[i].pos[X] = tgt_x[i];
                tgt[i].pos[Y] = tgt_y[i];
                tgt[i].no_enc = 0;
                tgt[i].no_to  = 0;
                tgt[i].no_nto = 0;
        }

        inptr = fopen("target.out","w");
        for (i=0; i<no_target; i++)
                fprintf(inptr, "%10.5f %10.5f \n", tgt[i].pos[X], tgt[i].pos[Y]
);
        fclose(inptr);
}

void init_ob()
/*------------------------------------------------------------------- */
/*    Function:  init_ob                                               */
/*    Parameters:                                                      */
/*    set the initial obstacle  position                               */
/*------------------------------------------------------------------- */
{
        int i, j, k;
        float rk_x[7000], rk_y[7000];
        float dm, ft, r, x1, x2, y1, y2;
        int   rk_pt, flg, done;
        float theta, th;
float rk[XY];
FILE *inptr;
/*                                      */
/*  Initialize the obstacle position */
/*                                      */

inptr = fopen("obst.dat","r");
j = 0;
while (fscanf(inptr, "%f %f", &x1, &y1) !=EOF)
{
        rk_x[j] = x1;
        rk_y[j] = y1;
        j++;
}
fclose(inptr);

ft = 0.3048;

for (i=0; i<no_rock; i++) {
    dm = 9.0 * ((float) drand48()) + 1.0;
    rk_pt = (int) (pi * dm ) + 1;
    r = dm * ft / 2.0;
    theta = twopi / rk_pt;

    done = 0;
    while (!done) {
        rk[X] = max_X_length * ((float) drand48());
        rk[Y] = max_Y_length * ((float) drand48());
        x1 = rk[X] - r - ft;
        x2 = rk[X] + r + ft;
        y1 = rk[Y] - r - ft;
        y2 = rk[Y] + r + ft;

        k = 0;
        flg = 0;
        while (!flg && (k<no_target)) {
            if ((tgt[k].pos[X] > x1) && (tgt[k].pos[X] < x2)) {
                if ((tgt[k].pos[Y] > y1) && (tgt[k].pos[Y] < y2))
                    flg = 1;
                else
                    flg = 0;
            }
            else
                flg = 0;
            k++;
        }
        if (flg)
            done = 0;
        else
            done = 1;
    }

    for (k=0; k<rk_pt; k++) {
        th = k * theta;
        rk_x[j] = r * cos(th) + rk[X];
        rk_y[j] = r * sin(th) + rk[Y];
        j++;
    }
}

no_obs = j;
```

```
        free(ob);
        if (!(ob = (struct obstacle *)malloc(no_obs * sizeof(struct obstacle)))
)
                fprintf(stderr, "bounce: malloc failed in init_ob\n");


        sort(rk_x, rk_y, no_obs);

        for (i=0; i<no_obs; i++)
        {
                ob[i].obpo[X] = rk_x[i];
                ob[i].obpo[Y] = rk_y[i];
        }


        inptr = fopen("obst.out","w");
        for (i=0; i<no_obs; i++)
                fprintf(inptr, "%10.5f %10.5f \n", rk_x[i], rk_y[i]);
        fclose(inptr);
}

void sort(float list_x[], float list_y[], int size)
{
        int out, in;
        float temp;

        for (out=0; out<size-1; out++)
            for (in=out+1; in<size; in++)
                if ((list_x[out] > list_x[in]) ||
                        (list_x[out] == list_x[in] && list_y[out] > list_y[in]))
)
                {
                    temp = list_x[in];
                    list_x[in] = list_x[out];
                    list_x[out] = temp;

                    temp = list_y[in];
                    list_y[in] = list_y[out];
                    list_y[out] = temp;
                }
}

void  wr_to_nto(int no_itr)
{
        div_t dv;
        char ch[13];
        int j, x, y;
        float ens, eto, ento;

        FILE *inptr;

        strcpy(ch, "panTO000.out");
        x = no_veh;

        if (x < 10 )
                ch[7] = ch[7] + x;
        else if (x < 100) {
                dv = div(x,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }
        else {
                dv = div(x,100);
                ch[5] = ch[5] + dv.quot;
                y = dv.rem;
                dv = div(y,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }

        inptr = fopen(ch,"w");
        for (j=0;j<no_target ;j++) {
                ens = (float) tgt[j].no_enc / (float) no_itr;
                eto = (float) tgt[j].no_to  / (float) no_itr;
                ento= (float) tgt[j].no_nto / (float) no_itr;
                fprintf(inptr, "%7.3f %7.3f %7.3f  %5d \n", ens, eto, ento,
        }
        fclose(inptr);
}
```

LIST OF REFERENCES

[1] Healey, A.J., Kim, J., "BUGS: Robot Control, UXO and Minefield Clearance", *Office of Naval Research Technical Report*, Naval Postgraduate School, Monterey, CA. December, 1996.

[2] Washburn, A.R., *Search and Detection*, Arlington, VA. ORSA Books, 1989.

[3] Healey, A.J., McMillian, S.M., Jenkins, D.A., McGhee, R.B., "BUGS: Basic UXO Gathering System", *Proc. Autonomous Vehicles in Mine Countermeasures Symposium*, Naval Postgraduate School, Monterey, CA. April, 1995.

[4] Healey, A.J., Kim, J., "Control of Small Robotic Vehicles in Unexploded Ordnance Clearance", *Proceedings IEEE ICRA 97*, Albuquerque, New Mexico. April, 1997.

[5] Jenkins, D.A., "BUGS: Basic Unexploded Ordnance Gathering System Effectiveness of Small, Cheap Robots", *MSME Thesis*, Naval Postgraduate School, Monterey, CA. June, 1995.

[6] Lewis, T.A., "Simulation of Small Robotic Vehicle Performance During UXO Gathering Operations Using Discrete-Event State-Space Control", *MSME Thesis*, Naval Postgraduate School, Monterey, CA. September, 1996.

DISTRIBUTION LIST

Defense Technical Information Center 8725, John J. Kingman Road, Ste 0944 Fort Belvoir, VA 22060-6218	2
Library, Naval Postgraduate School 411, Dyer Road Monterey, CA 93943-5101	2
Professor A. J. Healey, Code ME/Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
Mr. Chris O'Donnel Research and Development Department NAVEODTECHDIV 2008, Stump Neck Road Indian Head MD 20640	1
Mr. Chris DeBolt Research and Development Department NAVEODTECHDIV 2008, Stump Neck Road Indian Head MD 20640	1
Mr. Craig Freed Research and Development Department NAVEODTECHDIV 2008, Stump Neck Road Indian Head MD 20640	1
Mr. Tuan Nuygen Research and Development Department NAVEODTECHDIV 2008, Stump Neck Road Indian Head MD 20640	1
Naval Surface Warfare Center Coastal Systems Station 6307, West Highway 98 Panama City FL 32407-7001 Attn: Charles Bernstein, Code A62	3
Chris Hillenbrand Naval Undersea Warfare Center Newport, R. I., 02841-5047	1
Douglas Gage NCCOSC RDTE DIV. 531 San Diego CA 92152-7383	1

Naval Engineering Curricular Office (Code 34) Naval Postgraduate School Monterey, CA 93943-5002	1
Lieutenant Jack A. Starr 5674 Chico Way NW Bremerton, WA 93912	1
Dr. Tom Swean Office of Naval Research, Code 3210E 800, North Quincy Street Arlington, VA 22217	1